Image Motion Analysis using Inertial Sensors

Saunders, Thomas

Award date:
2015

Awarding institution:
University of Bath

Image Motion Analysis using Inertial Sensors

submitted by

Thomas Richard Saunders

for the degree of Doctor of Philosophy

of the

University of Bath

Department of Computer Science

June 2015

Summary

Understanding the motion of a camera from only the image(s) it captures is a difficult problem. At best we might hope to estimate the relative motion between camera and scene if we assume a static subject, but once we start considering scenes with dynamic content it becomes difficult to differentiate between motion due to the observer or motion due to scene movement. In this thesis we show how the invaluable cues provided by inertial sensor data can be used to simplify motion analysis and relax requirements for several computer vision problems.

This work was funded by the University of Bath.

Acknowledgements

I would like to offer my gratitude to those who have helped me complete this work. To my supervisors and colleagues, whose help, guidance and discussion has shaped the direction and content of this work. To my parents, who provided me with everything I needed and who enabled me to develop my interests in science and technology. And finally, to my wife, whose support and encouragement has kept me going throughout this long process.

Contents

Chapter 1

Introduction

In recent years digital imaging technology has advanced considerably. Cameras capable of capturing high resolution images and video have become an integral part of everyday life, and something that many people carry with them in the form of a mobile phone with built-in camera. As a result, computational photography is frequently used for a variety of tasks; image enhancement and stylisation, video stabilisation and face recognition as part of the auto-focus mechanism.

It is inevitable that these cameras might move whilst capturing data, whether intentionally, perhaps in the case of a camera panning as it tracks a moving object, or unintentionally, as a result of the force required to push the shutter button, causing the camera to shake.

In either case we end up with image data (either in the form of a sequence of images from one video or a single image) of a world that is moving relative to the optical sensor. If we have a sufficiently short exposure time relative to the magnitude of motion, then the individual image(s) will not reveal the motion of the camera, but if we find ourselves in a situation requiring a longer exposure time then the images will likely be degraded by the motion, resulting in blur.

Putting aside potential corruption due to blur, let's also consider how a moving camera complicates our understanding of the content of a scene. The image focused on the camera's sensor is constantly changing according to the motion of the camera, making it difficult to discriminate between static and moving regions of the scene as both are likely to be moving relative to the camera. The performance of tasks such as video stabilisation and deblurring is impeded by the presence of scene motion. Current techniques require costly motion analysis to mitigate these motions.

A moving object in a recorded video is likely to be the focus of the imagery, and the ability to isolate independent motion can be of great utility. Take for example the auto-focus mechanism of a camera: its goal is to decide which portion of the image (and ultimately what depth) to focus on, and having the option to find moving regions could benefit the process by identifying a high level feature. In sports broadcasting, overlays are often used to track a player or a ball as it moves across the pitch, often filmed by a moving camera.

The problem of understanding scene motion captured by a moving camera from an image pair or blurry image is difficult as the initial observation is that everything is moving. To solve this problem, we look for inspiration from the human visual system. The brain does not rely solely on visual information and prior learned knowledge of a scene; it senses the motion of the head and can effortlessly pick out moving objects even if the eyes themselves are moving. We find a parallel for this sensory feedback by mounting small inexpensive motion sensors on a camera. These sensors can give a good - but not perfect - insight into the forces acting on a camera, and as their size, cost and performance improve, they are becoming increasingly common in consumer devices such as mobile phones.

In this thesis we explore how inertial sensors can be utilised to better understand motion in several computer vision problems, thereby simplifying the process and making posible new applications that otherwise are difficult to realise or required extensive computation.

1.1 Research Hypothesis

The methods and results presented in this thesis develop the following assertion:

'Inertial sensors can simplify the task of analysing motion in moving-camera computer vision problems'.

1.1.1 Motivation

Inertial sensors are becoming ubiquitous, as are mobile devices that combine inertial sensors and a camera with a modest computer system. We believe there are several classes of computer vision problems that can realise significant benefits from utilising these sensors, leading to better solutions and/or more efficient processes.

The technology present in these mobile devices is advancing rapidly, to the point where operations on large images or video sequences are becoming the norm. However, many tasks, such as image deblurring, still present a challenge to the comparatively limited (when compared to a desktop machine) resources available. It is our belief that the data provided by inertial sensors can be put to use to allow otherwise insurmountable problems to be solved.

1.2 Outline and contributions of this thesis

Here we outline the subjects covered by this thesis and its contributions.

1.2.1 Time-alignment of inertial sensor data and video

In Chapter 2 we describe the theory and limitations of estimating camera motion using inertial sensors, and describe the design of our embedded inertial sending device. As part of this, we present a novel method for finding the globally optimal alignment of inertial data and a video sequence. We compare this method to the literature and show greatly improved robustness as a result of our use of dense motion analysis and ability to find the global minimum solution.

1.2.2 Optical flow confidence learned from inertial sensor data

In Chapter 3 we detail the problems involved with analysing motion in images using an optical flow field which is not always accurate. We work around this problem with our novel scheme for learning an optical flow confidence measure from real images and sensor data. This compares favourably with the state of the art methods as it relaxes the requirement of hand-labelled or synthetic ground truth training data. We compare confidence measures trained using both synthetic and real images, and show improved performance in the latter case when using real images as evaluation data. Our use of inertial sensors therefore allows for a confidence model that can be easily tailored to any environment / optical flow method, yielding improved confidence accuracy.

1.2.3 Image motion segmentation

Chapter 4 details a method for segmenting the non-rigid motion of multiple moving objects from a video using sensor data. By using a combination of the learned optical flow confidence measure and inertial sensor data we are able to develop a segmentation scheme that out-performs the current state-of-the art in terms of speed by a factor of nearly 50, whilst achieving an equivalent segmentation accuracy. In this chapter we analyse the performance of our method and show that its relative speed gain is a combination of both faster analysis and the ability to understand the motion in a quick-to-compute optical flow field.

1.2.4 Understanding scene-motion in blurred images

In Chapter 5 we describe the challenges associated with image deblurring and the additional complication presented by images corrupted by both camera-shake and scene motion. Our preliminary work demonstrates how a motion estimate from inertial sensors can isolate independent scene motion.

1.3 Mathematical Notation

Bold letters such as $\mathbf{I}$ are used to denote non-scalar quantities. Caligraphic letters such as $\boldsymbol{\mathcal{M}}$ are used to represent sets. Vector notation is used to represent images: $\mathbf{I} \in \mathbb{R}^N$ is an image with a total of $N = H\text{x}W$ pixels, where H and W are the height and width of the image respectively. A colour image with 3 channels of N pixels uses the same notation, even though there are effectively 3 times as many pixels. Unless otherwise specified, operations are performed on the grey-scale image. The element-wise product of two vectors is denoted by $\mathbf{a} \circ \mathbf{b}$.

Chapter 2

Camera Motion Estimation

This chapter introduces the concept of using hardware sensors to estimate the motion of a camera. A mathematical background is provided in section 2.1, section 2.2 gives a description of how physical motion can be inferred from sensors, and in section 2.3 we develop the model that relates the extrinsic camera motion to the captured images. Finally, in section 2.4, we give a detailed overview of the inertial sensing hardware and present a novel robust scheme for time-aligning captured images from a video with inertial sensor data.

It should be noted that in this work we assume the intrinsic properties of the camera, that is the lens, focal length, sensor size and focus distance, to remain fixed throughout each exposure. We assume the only changes to the camera to be its extrinsic parameters - the position and orientation.

2.1 Mathematical Background

The desired output of the motion sensing process is an estimate of the camera's position and orientation throughout a recorded sequence of images or single exposure. The position, $\mathbf{T}_i = [T_x \quad T_y \quad T_z]^T$, is a 3-vector comprised of the 3D camera position relative to an arbitrary origin in metres. The orientation of the camera with respect to gravity can be encoded by a rotation matrix $\mathbf{R}_i \in \mathbb{R}^{3x3}$. This form allows for a simple projection model that maps an homogeneous 3D point $\mathbf{z}$ to the corresponding 2D equivalent $\mathbf{x} = \mathbf{K}[\mathbf{R} \quad \mathbf{T}]\mathbf{z}$ as projected onto the image plane. This projective mapping is covered in more detail in section 2.3.

It is advantageous to also use a quaternion representation of orientation. Encod-

ing such a form requires just 4 scalars compared to the 9 of a 3x3 rotation matrix, and the numerical operations required to perform calculations are similarly reduced. This is of benefit when estimating orientation in real time on the embedded system, as memory and processing capacity are at a premium. In addition, a quaternion representation allows us to treat orientation prediction equations linearly and avoids the pit-fall of gimbal lock as it is free from singularities.

2.1.1 Quaternion Theory

In this section we state some background quaternion theory that will enable us to keep track of camera orientation and move between different frames of reference. A good introduction to the topic, as well as derivation of much of the following, can be found in [1].

A quaternion is a $\mathbb{R}^4$ vector of the form $\mathbf{q} = \begin{pmatrix} q_w & q_x & q_y & q_z \end{pmatrix}^T$. We can also write $\mathbf{q}$ in a compact form comprising scalar and vector parts $s = q_w$ and $\mathbf{v} = \begin{bmatrix} q_x & q_y & q_z \end{bmatrix}^T$ respectively:

$$q = \begin{pmatrix} s \\ \mathbf{v} \end{pmatrix} \tag{2.1}$$

Quaternion Conjugate The conjugate $\overline{\mathbf{q}}$ of a quaternion $\mathbf{q}$ is given by

$$\overline{\mathbf{q}} = \begin{pmatrix} q_w \\ -q_x \\ -q_y \\ -q_z \end{pmatrix} \tag{2.2}$$

Quaternion Product The product of two quaternions, $\mathbf{q}$ and $\mathbf{p}$, is written as $\mathbf{q} * \mathbf{p}$ and is found

$$\mathbf{q} * \mathbf{p} = \begin{pmatrix} q_w p_w - q_x p_x - q_y p_y - q_z p_z \\ q_x p_w + q_w p_x - q_z p_y + q_y p_z \\ q_y p_w + q_z p_x + q_w p_y - q_x p_z \\ q_z p_w - q_y p_x + q_x p_y + q_w p_z \end{pmatrix} \tag{2.3}$$

This operation is not commutative: $\mathbf{q} * \mathbf{p} \neq \mathbf{p} * \mathbf{q}$

Quaternion Norm The norm $\|\mathbf{q}\|$ of a quaternion $\mathbf{q}$ is given by

$$\|\mathbf{q}\| = \sqrt{\mathbf{q} * \overline{\mathbf{q}}} = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} \tag{2.4}$$

We use normalised quaternions to represent rotations. This can enforced by dividing each element by the norm:

$$\mathbf{q}' = \frac{\mathbf{q}}{\|\mathbf{q}\|} \tag{2.5}$$

Rotation of a vector We can rotate a vector $\mathbf{v} \in \mathbb{R}^3$ by a unit quaternion $\mathbf{q}$ with

$$\mathbf{v}' = \mathbf{q} * \begin{pmatrix} 0 \\ \mathbf{v} \end{pmatrix} * \overline{\mathbf{q}} \tag{2.6}$$

The rotation can be applied in reverse by first conjugating $\mathbf{q}$.

Unit Quaternion to Rotation Matrix The rotation matrix $\mathbf{R}$ corresponding to a unit quaternion $\mathbf{q}$ can be found using:

$$\mathbf{R} = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y + q_w q_z) & 2(q_x q_z - q_w q_y) \\ 2(q_x q_y - q_w q_z) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z + q_w q_x) \\ 2(q_x q_z + q_w q_y) & 2(q_y q_z - q_w q_x) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix} \tag{2.7}$$

Quaternion Integration A quaternion representation of orientation $\mathbf{q}$ can be updated according to an angular velocity $\boldsymbol{\omega}$ by integrating over a timestep Δt. This is equivalent to solving the following first order differential equation:

$$\dot{\mathbf{q}} = \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\mathbf{q} \tag{2.8}$$

where

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix} = \begin{bmatrix} -\lfloor \boldsymbol{\omega} \rfloor & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^T & 0 \end{bmatrix} \tag{2.9}$$

and $\lfloor \boldsymbol{\omega} \rfloor$ is the skew-symmetric matrix operator that computes a cross-product for $\boldsymbol{\omega}$:

$$\lfloor \boldsymbol{\omega} \rfloor = \begin{bmatrix} 0 & \omega_z & -\omega_y \\ -\omega_z & 0 & \omega_x \\ \omega_y & -\omega_x & 0 \end{bmatrix} \tag{2.10}$$

If the angular velocity $\boldsymbol{\omega}$ is assumed to be constant over the the integration period Δt then a zeroth order quaternion integrator[1] may be used to yield a quaternion $\mathbf{q}'$ representing the relative rotation:

$$\mathbf{q}' = \begin{pmatrix} \cos\left(\frac{|\boldsymbol{\omega}|}{2}\Delta t\right) \\ \frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|}\sin\left(\frac{|\boldsymbol{\omega}|}{2}\Delta t\right) \end{pmatrix} \tag{2.11}$$

2.2 Motion Sensing

A crucial component of this work is a means of estimating the motion of a camera whilst it records a video or captured an image. This could be achieved either by externally monitoring how the camera moves, or by internally sensing the movement of the camera relative to its environment.

In this thesis we cover two types of image motion. Firstly videos, in which the single un-blurred frames are captured at evenly spaced intervals throughout the video. In this case, we're interested in the position and orientation of the camera at the time of each exposure - we only want the relative motion between frames. For blurry images, the 'journey' of the camera is vital for recovering the true motion of the camera as the blurred image is essentially just the average of a stack of images of the world as seen by the camera, which we approximate as being images captured at discrete intervals.

2.2.1 External Motion Capture

Motion capture is a common technique for recording the 3D motion of human actors for animating 3D models. Known objects, typically small spheres, are attached to specific locations on a moving subject. These are then tracked by an array of cameras, and multiple view geometry techniques used to reconstruct the 3D position and motion of each point.

Köhler *et al.* [2] used the technique to capture the motion of a camera with a

series of markers attached on the end of sticks whilst it captured a blurry image. The obvious advantage of estimating camera motion in this way is the accuracy of measurements; they're taken relative to fixed reference points, so the estimates do not accumulate error over time. Such an approach also necessitates strict control over the scene, restricts the possible field of view of the camera and requires an array of specialist equipment. This sort of setup might be appropriate in a studio but would have limited applicability for casual use.

2.2.2 Internal Motion Sensing

Internal motion capture can be achieved by rigidly attaching some sort of sensor to the camera, such that the sensor experiences the camera motions. When it comes to measuring motion, it is important to consider that a rotation about one point on the camera will cause both a rotation and translation at any other point. We are predominantly interested in the motion of the image sensor, but the reality is that it is not practical to have motion sensors located at the optical centre.

Providing we firmly mount our sensors to the camera, we can consider the pair to be a single rigid body. Therefore, the orientation with respect to the world frame is the same at both the image plane and motion sensor, with only the relative translation of each point with respect to the world varying.

The requirements for the sensor platform are:

- it must be small and unobtrusive such that it does not impede the usual operation of the camera
- its motion data must be synchronised with the camera's imagery
- it must record motion to non-volatile memory for off-line analysis

Micro-electromechanical systems (MEMS), such as a gyroscope or accelerometer, are a common feature in many modern gadgets for a variety of applications: landscape / portrait mode switching, drop detection and game interaction amongst others. They are an attractive solution for this project as they're small (in the region of 5x5mm) and inexpensive.

The sensor data is recorded in the world coordinate system (x_w, y_w, z_w) and later transformed into the camera coordinate system (x_c, y_c, z_c) when considering image data. These coordinate systems are outlined in figure 2-1.

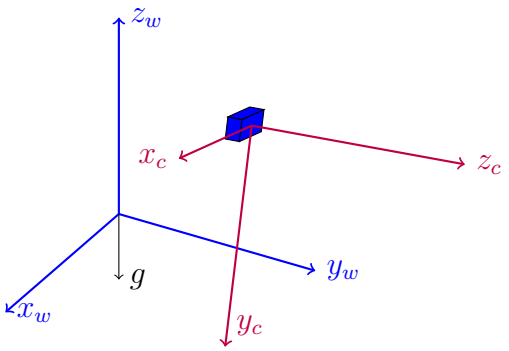


Figure 2-1: World (blue) and Camera (red) coordinate systems. The blue box represents the camera, and **g** is the gravity vector

2.2.3 Accelerometer

An accelerometer measures the acceleration $\mathbf{a} \in \mathbb{R}^3$ in ms^{-2} acting on the device. For a static object, the forces measured will be as a result of gravity:

$$\mathbf{a} = \mathbf{R}_{(\mathbf{q})}\mathbf{g} = \mathbf{R}_{(\mathbf{q})} \begin{bmatrix} 0 & 0 & -9.81 \end{bmatrix}^T ms^{-2} \tag{2.12}$$

where $\mathbf{R}_{(\mathbf{q})}$ is a rotation matrix that rotates from the world frame to the accelerometer frame according to the current orientation $\mathbf{q}$. Only in free-fall can the accelerometer measure 0 in all axes. The classical construction of an accelerometer is a damped mass on a spring, which is either linearly compressed or stretched as a result of acceleration. The displacement of the mass is used to determine the forces acting upon it.

MEMS accelerometers operate in a similar manner except on a much smaller scale. Typically these devices have a mass on the end of a cantilever which deflects with acceleration. The displacement can be estimated by measuring the capacitance between a fixed beam and the cantilever beam with the distance between the two plates varying as the mass moves.

The accelerometer we used had a configurable 16 bit output range of either $\pm 2g, \pm 4g, \pm 8g$ or $\pm 16g$. In each case the full 16 bit range is scaled such that in $\pm 2g$ mode, $-2g = -32768$ and $+2g = 32767$. We used the accelerometer in $\pm 2g$ output as this gave the greatest sensitivity of 16384 LSB/g.

If the device is moving then the force experienced, $\mathbf{a}$, will be a combination of translational acceleration $\mathbf{a}_t$ (caused by translation of the camera or off-centre

rotation), and gravitational acceleration $\mathbf{a}_g$:

$$\begin{aligned} \mathbf{a} &= \mathbf{a}_t + \mathbf{a}_g & (2.13) \\ &= \mathbf{a}_t + \mathbf{R}_{(\mathbf{q})}\mathbf{g} & (2.14) \end{aligned}$$

Measurement Model

We model the data output by the accelerometer, $\mathbf{a}_m$, as follows:

$$\mathbf{a}_m = \mathbf{a} + \mathbf{b}_a + \mathbf{n}_a \tag{2.15}$$

where $\mathbf{a}$ is the true acceleration, $\mathbf{b}_a$ is accelerometer bias and assumed to be constant throughout the short duration of our videos and images, and $\mathbf{n}_a$ is accelerometer measurement noise, assumed to be white Gaussian.

Here we have a problem; how can we tell if the acceleration being measured by the device is due to translation or rotation (with centre of rotation not at the origin of the accelerometer) of the sensor?

Orientation from acceleration data If we assume the device to be stationary or travelling at a constant velocity, then the measured acceleration will be the result of gravity only:

$$\mathbf{a} = \mathbf{R}_{(\mathbf{q})}\mathbf{g} \tag{2.16}$$

In the long term, an accelerometer can be used to provide a fixed reference point for measuring the orientation, assuming there are sufficient periods with steady velocity.

Translation from acceleration data Let's assume we have an acceleration $\mathbf{a}_t$ that is the result of translation only. We can use this to update the velocity of the device, and subsequently its displacement. We have a system of differential equations:

$$\begin{aligned} \dot{\mathbf{v}}_n &= \mathbf{a}_n = \mathbf{a}_m - \mathbf{b}_a - \mathbf{R}_{(\mathbf{q})}\mathbf{g} & (2.17) \\ \dot{\mathbf{T}}_n &= \mathbf{v}_n & (2.18) \end{aligned}$$

The position and velocity of the camera, $\mathbf{T}_n$ and $\mathbf{v}_n$, can be updated for each

timestep:

$$\mathbf{v}_n = \mathbf{v}_{n-1} + \dot{\mathbf{v}}_n \Delta t \tag{2.19}$$

$$\mathbf{T}_n = \mathbf{T}_{n-1} + \mathbf{v}_n \Delta t \tag{2.20}$$

where $\mathbf{v}_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$, $\mathbf{T}_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$. The problem with this approach is that it is completely open-loop; small errors in $\mathbf{a}_m$ accumulate in $\mathbf{v}_n$ which very quickly cause $\mathbf{T}_n$ to drift, and we have no way of recovering. The use of GPS as a feedback to the system to reduce drift would be of limited value in this project as the magnitude of likely camera translation is far less than the precision of consumer GPS data for a typical recorded video or image.

As the accelerometer measures the rate of change of velocity, it is crucial to initialise velocity correctly if we want to track displacement. If we assume $\mathbf{v}_0$ to be zero when really the device is moving by $2ms^{-1}$, the estimate of $\mathbf{T}_n$ will drift by 2 metres every second. In light of this, position estimation over long periods of time without some sort of feedback measurement is impractical. As a result, we can only use the acceleration data for estimating translation when we're able to make assumptions about the initial conditions of the camera and the period of estimation is short (less than a second), or where feedback is available.

Calibration We use gravity as a known force to calibrate the accelerometer. We mount the inertial system on a level surface with each axis alternately oriented up and down to measure the average in both over a period of 1 second to get $\mathbf{a}_{\text{up}}$ and $\mathbf{a}_{\text{down}}$. We then determine the bias $\mathbf{b}_a$:

$$\mathbf{b}_a = \frac{\mathbf{a}_{\text{down}} - \mathbf{a}_{\text{up}}}{2} \tag{2.21}$$

2.2.4 Gyroscope

A gyroscope is a device used to measure the change in orientation of an object. Mechanical gyroscopes consist of a spinning disc mounted in a gimbal. As the gyroscope's orientation changes, the rotating disc's inertia acts to oppose the motion, causing its deflection to be much less than the rest of the gimbal, effectively acting as a fixed reference from which the change in orientation can be estimated.

Miniaturised versions of these devices operate by oscillating a tiny plate that gets deflected by the Coriolis effect as the package rotates. The deflection is measured by a set of capacitive plates, which is used to compute the angular rate. They offer considerable advantages over the traditional mechanical options, due to their small size, low power consumption and low cost, whilst performing a similar function.

The digital gyroscope we used outputs 16bit signed integer data. The sensitivity range can be configured from ±250 to ±2000 degrees per second, in each case spreading the full 16bits across the range. We opted for the smallest range as we did not anticipate camera rotation speeds beyond this, and this allowed for a greater precision (131 least significant bits per degree per second).

Measurement Model We model how the measured angular velocity from the gyroscope, $\boldsymbol{\omega}_m$, relates to the true angular velocity $\boldsymbol{\omega}$ as follows:

$$\boldsymbol{\omega}_m = \boldsymbol{\omega} + \mathbf{b}_\omega + \mathbf{n}_\omega \tag{2.22}$$

where $\mathbf{b}_\omega$ is a constant gyroscope bias and $\mathbf{n}_\omega$ is Gaussian white noise.

The data provided by the gyroscope helps us estimate a change in orientation from one time step to the next. The orientation estimate $\mathbf{Q}_{n-1}$ at time t_{n-1} is updated to get the orientation $\mathbf{Q}_n$ at time t_n, using

$$\mathbf{Q_n} = \mathbf{q}' * \mathbf{Q}_{n-1} \tag{2.23}$$

where $\mathbf{q}'_n$ is a quaternion representing the relative rotation and is the result of integrating $\boldsymbol{\omega}$ over the time interval, as defined in equation 2.11.

In this way, we can estimate relative orientation change, starting from $\mathbf{Q_0} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}^T$, for each reading of the gyroscope. We can find the relative change between the orientation at t_n and t_{n+X}, by conjugating the reference quaternion $\mathbf{Q}_n$:

$$\vec{\mathbf{Q}_\mathbf{r}} = \overline{\mathbf{Q}}_n * \mathbf{Q}_{n+X} \tag{2.24}$$

Estimating orientation in this manner is highly sensitive to noise in the angular rate data. As it is effectively an open-feedback system, it has no means of correcting the estimate, so will accumulate error as time progresses, drifting from the true value. Whether or not this is a problem depends on the signal to noise ratio of

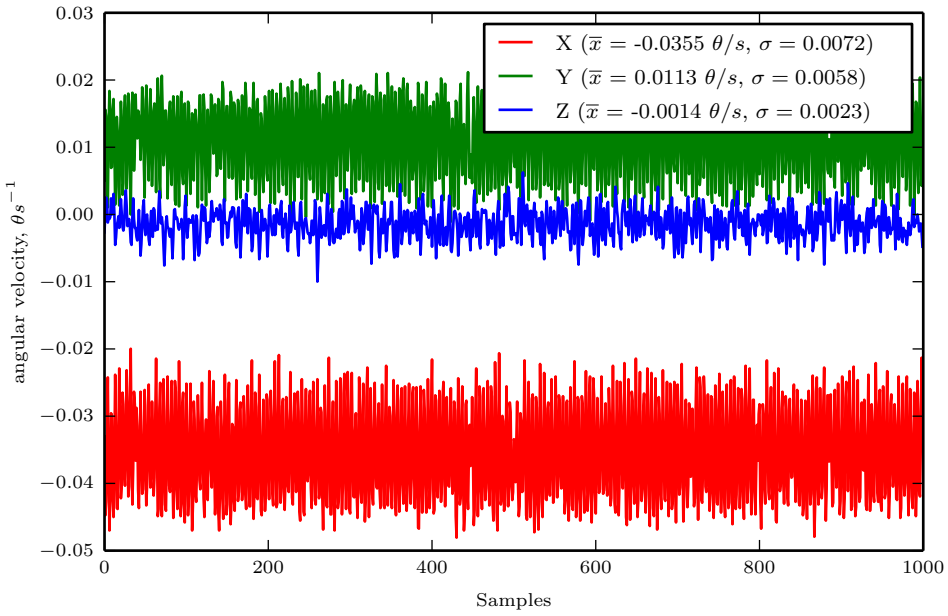
actual camera motion versus accumulated noise over the period of measurement. The frame-to-frame motion estimate of a video can be quite accurate as the interval between frames is generally small, but if considering the overall motion of an entire video, the drift will be considerable by the end of the sequence.

Calibration The angular velocity readings from the gyroscope are integrated, so any error in the system quickly builds up, therefore it is essential to perform calibration. MEMS Gyroscopes are susceptible to a temperature varying offset bias $\mathbf{b}_\omega$; whilst stationary, the readings will likely be non-zero. We counter this with a short self-calibration step that is performed each time the device is powered up. The gyroscope, assumed to be static, records the angular velocities $\boldsymbol{\omega}_m$ over a period of several seconds as shown in figure 2-2a. The bias $\mathbf{b}_\omega$ is then set as the average value in each axis and is subtracted from future measurements according to the measurement model in equation 2.22. We assume the gyroscope biases to remain constant throughout the duration of recording, hence $\dot{\mathbf{b}}_\omega = 0$.

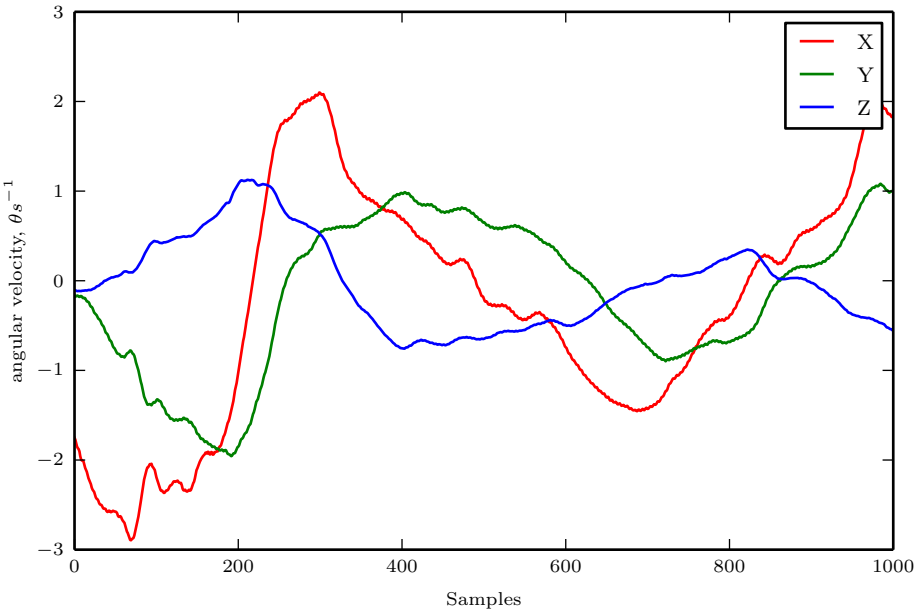
Linearity Verifying the angular response of the gyroscope requires rotating it at known speeds. We achieved this by attaching the sensor system to a plate that was then rotated by a stepper motor (see figure 2-4). Because the stepper motor works by making discrete steps rather than a continuous motion we averaged the angular velocity readings to produce a mapping of the angular response of the gyroscope, as shown in figure 2-3. We found the measurements to be consistently linear throughout the range we were interested in. Rotation speeds beyond $\pm$ 50 degrees per second were not tested due to limitations of the test platform.

Magnetometer

A magnetometer measures the strength of a magnetic field. If measuring a constant magnetic field, such as the Earth's, then the field may be used as a fixed point of reference, and can be used to infer changes in orientation. A magnetometer provides data that, whilst low frequency compared to the inertial sensors previously discussed, is still very useful for avoiding long term accumulation of error - assuming the magnetic field it is measuring does not change. The last point is important because any ferrous materials moving in the vicinity of the magnetometer are likely to influence its measured electromagnetic field. In the context of this work we found

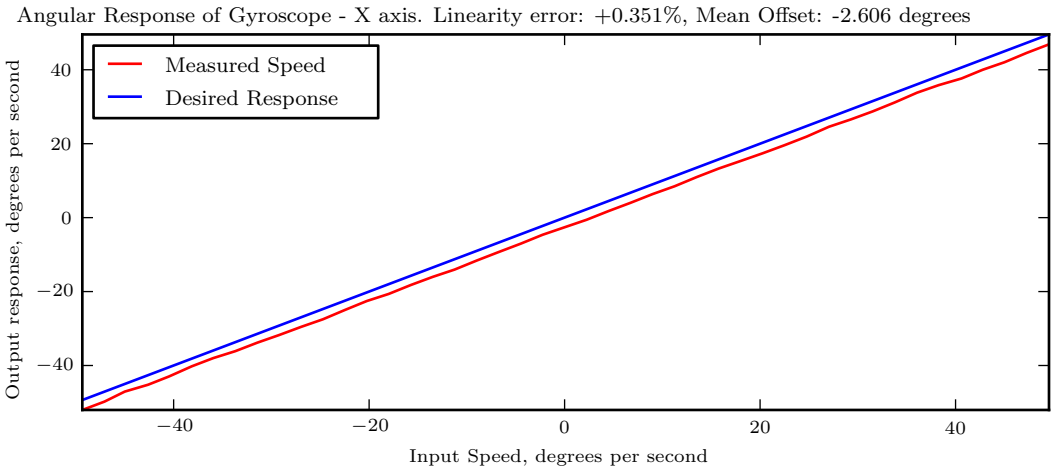


(a) Stationary

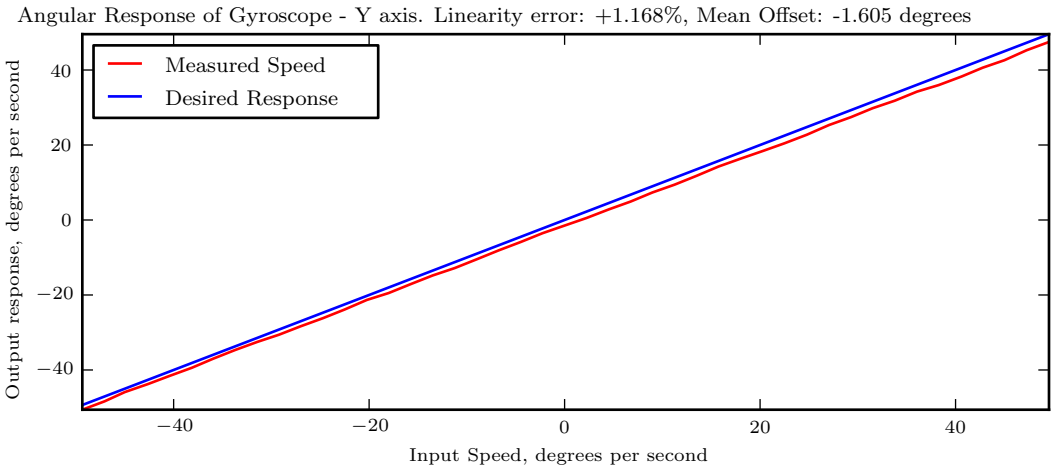


(b) Moving

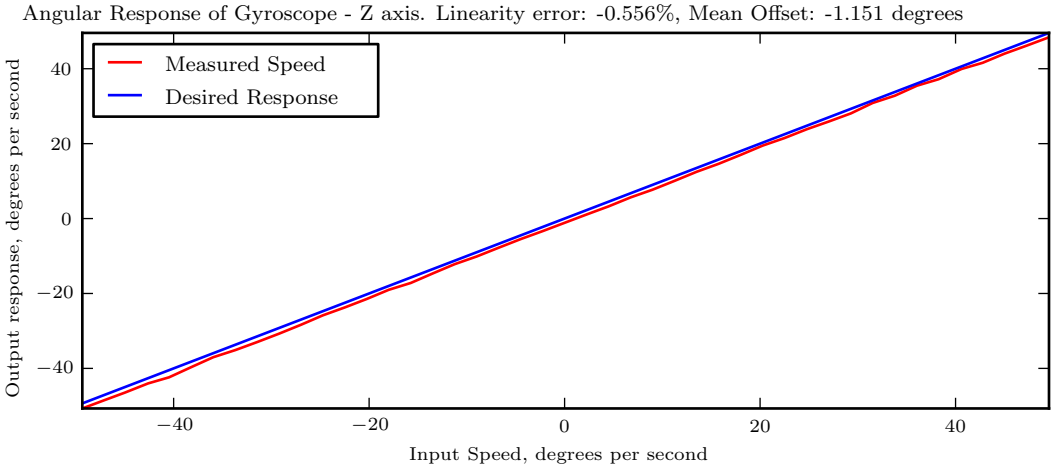
Figure 2-2: Raw data from moving and static gyroscope. Static plot (a) includes mean and standard deviation. The z axis noise is less due to the physical construction of the gyroscope



(a) X axis



(b) Y axis



(c) Z axis

Figure 2-3: Validation of Gyroscope performance using rotating platform

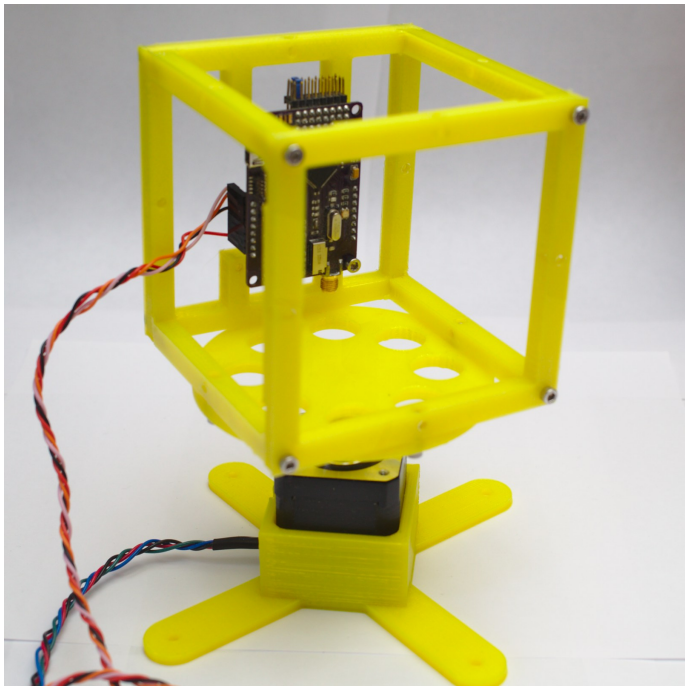


Figure 2-4: 3D printed turntable for validating angular velocity response of gyroscope. The stepper motor mounted in the base rotates the platform back and forth. The cube can be oriented to change the axis under evaluation

it unnecessary to use the magnetic data as the images from the camera provide similar feedback.

2.2.5 Sensor Fusion

Up until this point we have covered the individual aspects of gyroscopes and accelerometers in detail. Each has its limitations; a gyroscope is susceptible to drifting over time, whilst an accelerometer struggles to differentiate between acceleration due to translation, rotation or gravity. If the data from the two sensors is combined then we can significantly improve our understanding.

A gyroscope is very good at detecting high frequency changes in attitude and ensures that the orientation estimate responds quickly to rotations - and the data it provides is accurate in the short term. The accelerometer is unsuitable for estimating short term orientation changes, but in the long term it can be used to ensure that significant drift in orientation caused by the gyroscopes does not occur.

It can take some time for the initial orientation estimate to converge on the actual orientation, depending on the gains in the system, which specify how much the data from the one sensor should be trusted relative to the others. If it takes even half a second to converge then, the estimate of absolute orientation at the beginning of a recorded sequence would be unreliable. This has the consequence of making estimation of translation acceleration, and hence position tracking, difficult. Joshi *et al.* [3] make the assumption that measured acceleration is normally distributed about gravity, and estimate the gravity vector as the mean of acceleration over the course of a blurry exposure. We avoid making such assumptions by the design of our inertial system, and overcome this problem by continuously computing an estimate of orientation on the embedded system, so we are able to provide an estimate of orientation as soon as we start recording data. More details of the implementation of our sensors are provided in section 2.4.

2.3 Predicting Image Motion from Sensors

If we can predict how we expect the content of an image to move given what the sensors have told us, and we can observe how the an image actually moves via optical flow, then we can learn about the discrepancies between the two which arise

as a result of scene motion or errors in the camera motion estimation. Firstly we will cover how our estimate of camera motion translates to motion captured on the image sensor.

2.3.1 Image Model

The intensity of light from a world point $\mathbf{z} = (z_x, z_y, z_z)$ at an instantaneous time t is captured by the image sensor at $\mathbf{x} = (x_1, x_2)$. The location of the focused point source on the image plane is a function of the camera's projection matrix, and, in homogeneous coordinates, can be written as:

$$\mathbf{x}(t) = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \mathbf{P}(t) \begin{bmatrix} z_x \\ z_y \\ z_z \\ 1 \end{bmatrix} \tag{2.25}$$

For a moving camera, $\mathbf{P}(t)$, which is referred to as the projection matrix, will vary at each timestep. This causes a fixed world point $\mathbf{z}$ in 3D space to be projected to different image plane coordinates $\mathbf{x}(t_1), \mathbf{x}(t_2)$ at times t_1 and t_2. $\mathbf{P}(t)$ is a 3x4 matrix and can be decomposed into parts representing the intrinsic parameters of the camera $\mathbf{K}$ (see section 2.3.2), and the time-varying extrinsic components, the rotation $\mathbf{R}_c(t)$ and translation $\mathbf{T}_c(t)$ of the camera [4, p. 156]:

$$\mathbf{P}(t) = \mathbf{K} \begin{bmatrix} \mathbf{R}_c(t) & \mathbf{T}_c(t) \end{bmatrix} \tag{2.26}$$

If we are to use the inertial sensor data to detect motion in the corresponding images, we must transform it from the inertial frame to the camera frame, finding the motion at the image sensor. If $\mathbf{q}_w$ is the orientation from the sensors in world coordinates, and ${}^w\vec{\mathbf{q}}_c$ is a rotation from world to camera frame, then we transform the motion estimates:

$$\mathbf{q}_c = {}^w\vec{\mathbf{q}}_c * \mathbf{q}_w * \overline{{}^w\vec{\mathbf{q}}_c} \tag{2.27}$$

$$\mathbf{T}_c = \mathbf{R}_{({}^w\vec{\mathbf{q}}_c)}(\mathbf{T}_w + \mathbf{u}_c - \mathbf{R}^T_{(\mathbf{q}_w)}\mathbf{u}_c) \tag{2.28}$$

where $\mathbf{u}_c$ is a vector from the sensor platform to the image sensor, and is used to

transform rotations experienced by the sensors to a combination of translation and rotation at the optical centre.

In this thesis we are dealing with unknown scenes with unknown depth. We assume scenes to be planar, or to be sufficiently far away from the camera to avoid significant parallax, or for the only significant camera motion to be rotation. This enables us to model image motion with affine transformations, using an homography matrix $\mathbf{H}$ that operates in 2D image space. The homography describes the invertible transformation between two planes and is constructed differently to the projection matrix:

$$\mathbf{H} = \mathbf{K}\left(\mathbf{R} + \mathbf{T}\begin{bmatrix}0 & 0 & 1\end{bmatrix}\right)\mathbf{K}^{-1} \tag{2.29}$$

This is a common form used in the literature, for example in [3, eqn. (4)]. A 2D point $\mathbf{p} = [p_1 \quad p_2 \quad 1]^T$ in image coordinates can then be transformed via $\mathbf{H}$:

$$\mathbf{p}' = \mathbf{H}\mathbf{p} \tag{2.30}$$

Sensor-Predicted Flow Field

If $\mathbf{R}_i$ and $\mathbf{T}_i$ are the rotation and translation at frame i, and $\mathbf{R}_{i+1}$ and $\mathbf{T}_{i+1}$ are at frame $i+1$, then we can estimate the relative motion of a pixel between frames i and $i+1$:

$$\Delta\mathbf{p} = \mathbf{p} - \mathbf{K}\left(\mathbf{R}_{i+1}\mathbf{R}_i^T + (\mathbf{T}_{i+1} - \mathbf{T}_i)\begin{bmatrix}0 & 0 & 1\end{bmatrix}\right)\mathbf{K}^{-1}\mathbf{p} \tag{2.31}$$

We consider the sensor flow field, denoted $\mathbf{s} \in \mathbb{R}^{2N}$, to be the predicted relative motion for each pixel i in an image:

$$\mathbf{s}_i = \Delta\mathbf{p}_i \tag{2.32}$$

Observed Optical Flow Field

Optical flow is an approximate measure of the apparent motion between two images, represented as a two-component vector of relative motion for each pixel. An ideal optical flow field describes the motion of a scene feature as rendered on the discrete image plane from one image to the next. A common technique is to track pixel

intensities between frames:

$$I(\mathbf{x}_i, t) = I(\mathbf{x}_i + \mathbf{u}_i, t + 1) \tag{2.33}$$

where $I(\mathbf{x}_i t)$ is image intensity as a function of 2D coordinates $\mathbf{x}_i \in \mathbb{R}^2$ and time t, and $\mathbf{u}_i \in \mathbb{R}^2$ is the 2D velocity at $\mathbf{x}_i$.

This intensity constraint can work well for pixels with distinguished structures, but fails when handling uniformly textured regions. The situation can be improved by a global smoothing of the flow field that penalises sharp variations in $\mathbf{u}$ in small windows. The end result is a more uniform flow field at the expense of smoothing the boundaries between patches of discontinuous motion.

Popular optical flow methods include Lucas-Kinade [5] and Horn-Schunck [6]. The actual method used to estimate the optical flow is not of great significance to this work, as we note the choice can depend on several factors, such as efficiency, memory requirements, accuracy and availability of an implementation. In this work we use the method of Farneback *et al.* [7].

2.3.2 Camera Calibration

The camera calibration matrix $\mathbf{K} \in \mathbb{R}^{3\text{x}3}$ encodes the properties that are intrinsic to a particular combination of lens and imaging sensor, the sum of which we refer to as 'the camera'. We use a simple ideal pinhole camera model that neglects defects such as barrel distortions. Under this model we construct $\mathbf{K}$ as:

$$\mathbf{K} = \begin{bmatrix} fs_x & fs_\theta & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.34}$$

where f is the focal length of the lens in metres, s_x and s_y are the pixel densities in pixels per metre, s_θ is the skew, which we assume to be 0 corresponding to orthogonal x and y axes, and, finally, o_x and o_y correspond to the offset in pixels of the centre of the image sensor.

The images and videos captured by our camera contain EXIF metadata. From this, we can obtain the focal length and focal plane resolution in each axis, and total size of sensor in pixels. If an image has been resized, for example when pixel binning is employed by the sensor to decrease the resolution in video mode, then s_x and s_y

must be adjusted. We note that although this camera model greatly simplifies the true optical response of the system, and uses the focal length f as reported by the camera which is not always exact, it is sufficient for our needs as we are already using noisy approximate motion estimates from the sensors, and one of the main goals of this thesis is to demonstrate how inertial measurements can simplify the solution to computer vision problems, relaxing any requirement for a precise camera model.

2.3.3 Sensor-Image motion correlation

We have an estimate of image motion $\mathbf{s}$, and wish to compare it to the motion observed by the camera. The observed image motion $\mathbf{u} \in \mathbb{R}^{2N}$ is the relative motion vector at each pixel experienced by the camera between τ_1 and τ_2. For a video this is the optical flow field between a pair of adjacent frames $\mathbf{I}_i$, $\mathbf{I}_{i+1} \in \mathbb{R}^N$ captured at τ_1 and τ_2 respectively. For a blurry image $\mathbf{u}$ is the end points of the blur function at each pixel for an exposure that runs from τ_1 to τ_2.

Assuming our simplified camera model holds true and our predicted sensor flow $\mathbf{s}$ (found as the relative motion between τ_1 and τ_2) is free from imperfections, then we would find that $\mathbf{u} - \mathbf{s} = 0$. In reality this is unlikely for several reasons: inaccurate estimation of camera motion yielding errors in $\mathbf{s}$, imperfect estimation of observed motion $\mathbf{u}$ (perhaps as a result of homogeneous regions with little texture), noise in the image capture process, errors as a result of simple camera model, scene motion and so on. We now investigate how these errors manifest in $\mathbf{s}$ and $\mathbf{u}$, whilst assuming the scene to be static.

Translational camera motion will result in a greater magnitude of $\mathbf{u}$ for objects closer to the camera than those further away, much in the same way as using a calibration $\mathbf{K}$ computed for a larger focal length f (compared to the true value) increases the motion magnitude. Errors in the estimation of camera motion will result in divergence from the true trajectory.

We found that by splitting the relative motion fields into two components - magnitude and direction - we were better able to understand any differences. We compensated for an observed error margin proportional to motion magnitude with a weighting term $\boldsymbol{\mu}$ for each pixel that reduces the penalty for differences between $\mathbf{u}$ and $\mathbf{s}$ as the motion magnitude increases:

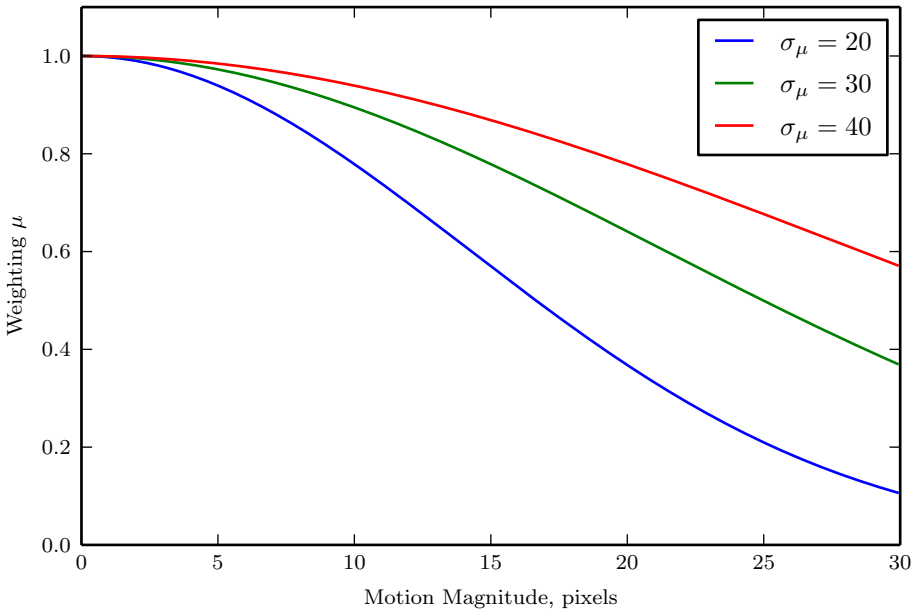


Figure 2-5: Motion magnitude compensation μ allows for increased error between $\mathbf{s}$ and $\mathbf{u}$ as the magnitude of optical flow increases. The variable σ_μ controls this falloff, with lower values used to compensate for noisier systems

$$\boldsymbol{\mu} = \exp\left(-\left(\frac{\|\mathbf{s}\|}{\sigma_\mu}\right)^2\right) \tag{2.35}$$

Magnitude Correlation

The magnitude correlation $\mathbf{c}_m \in [0, 1]$ is a measure of how well the predicted motion magnitude $\|\mathbf{s}\|$ matches the observed image motion magnitude $\|\mathbf{u}\|$. As the overall magnitude increases, the absolute difference becomes less important, so we find the ratio of difference to the sum of observed and estimated motion magnitudes for each pixel:

$$\boldsymbol{\rho} = \frac{\|\mathbf{s}\| - \|\mathbf{u}\|}{\|\mathbf{s}\| + \|\mathbf{u}\|} \tag{2.36}$$

The magnitude correlation $\mathbf{c}_m$, shown in figure 2-7, is then:

$$\mathbf{c}_m = \exp\left(-\left(\frac{\boldsymbol{\mu}\boldsymbol{\rho}}{\beta}\right)^2\right) \tag{2.37}$$

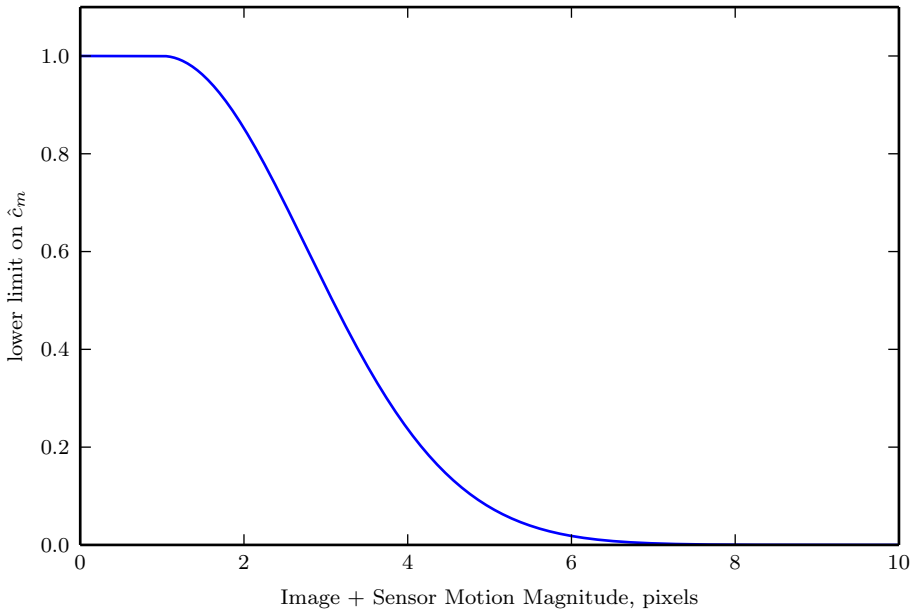


Figure 2-6: $\tilde{\mathbf{c}}_m$ (equation 2.38) is the lower limit on motion correlation, preventing noise in $\mathbf{u}$ and $\mathbf{s}$ dominating the result in areas where motion magnitude is low

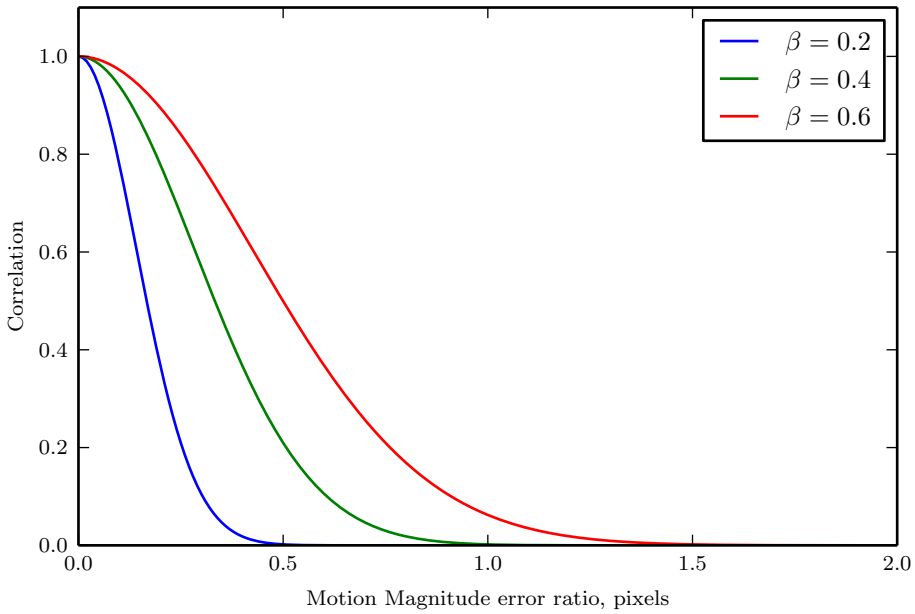
In regions with low motion magnitude, the ratio of optical flow error to $\|\mathbf{s}\|+\|\mathbf{u}\|$ is quite high, which can result in a low correlation. We compensate for this by setting a lower limit on $\mathbf{c}_m$ that drops off as the magnitude of motion increases (see figure 2-6), such that $\hat{\mathbf{c}}_m = \max(\mathbf{c}_m, \tilde{\mathbf{c}}_m)$, with the lower limit found using:

$$\tilde{\mathbf{c}}_m = \exp\left(-\frac{\max(\|\mathbf{s}\| + \|\mathbf{u}\| - 1, 0)^2}{\sigma_l}\right) \tag{2.38}$$

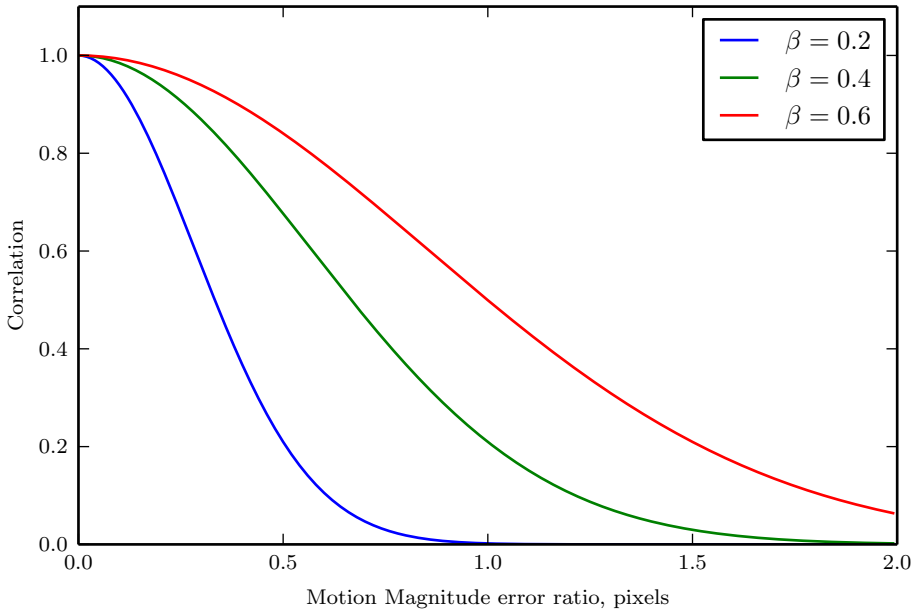
The effect of this compensation factor is shown in figure 2-11d, where the motion magnitude is very low, so an error of one or two pixels is insignificant as it is likely to be noise. The result is that the correlation field $\mathbf{C}$ is set to strong correlation in most regions, apart from where the ball is moving.

Direction Correlation

The motion directions, $\mathbf{u}_\theta$ and $\mathbf{s}_\theta$ corresponding to the direction of motion in the observed image and estimated sensor motion fields respectively, are found as the angle of the vector:



(a) Low Motion Magnitude



(b) High Motion Magnitude

Figure 2-7: Motion magnitude correlation with both low and high motion magnitude. The error ratio is the absolute difference in magnitude of **u** and **s** relative to the magnitude of **u**. The tolerance of errors between **u** and **s** increases as μ decreases from (a) to (b)

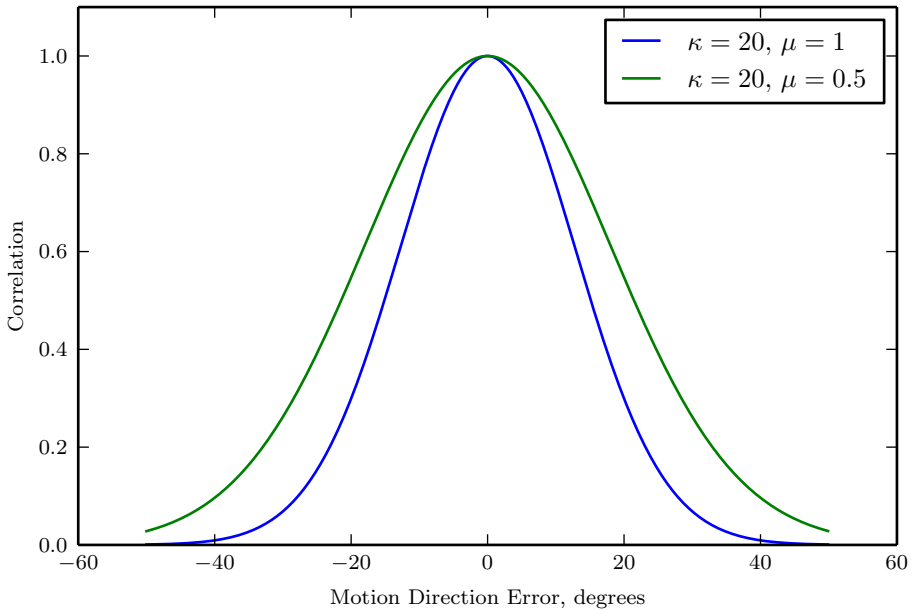


Figure 2-8: We relax the penalty for poor correlation between directions of $\mathbf{u}$ and $\mathbf{s}$ as the magnitude of $\mathbf{u}$ increases (causing μ to decrease, as per figure 2-5)

$$\mathbf{s}_\theta = \text{atan2}(\mathbf{s}_{\theta,y}, \mathbf{s}_{\theta,x}) \tag{2.39}$$

with the differences between $\mathbf{u}_\theta$ and $\mathbf{s}_\theta$ assumed to be uni-modally distributed over a circle. We model these differences using a von Mises distribution, which works favourably in our implementation as an angle difference of 359° is rightly handled the same as a difference of just 1°. The direction correlation $\mathbf{c}_d \in [0, 1]$ is reduced as the difference between $\mathbf{s}_\theta$ and $\mathbf{u}_\theta$ increases, as shown in figure 2-8. The penalty function takes the form:

$$\mathbf{c}_d = \exp\left(\kappa \boldsymbol{\mu} \left(\cos\left(\mathbf{s}_\theta - \mathbf{v}_\theta\right) - 1\right)\right) \tag{2.40}$$

It is not viable to draw any conclusion from the differences in $\mathbf{s}_\theta$ and $\mathbf{u}_\theta$ in regions with low image motion magnitude because the motion direction is mostly noise. If there is little difference between $\|\mathbf{u}\|$ and $\|\mathbf{s}\|$ in such areas, then the motion correlates well, and we do not want poor direction estimates in $\mathbf{u}_\theta$ to impede this. We set a lower limit on the value of $\mathbf{c}_d$ (of the same form as shown in figure 2-6) that is related to the magnitude of image motion, such that $\hat{\mathbf{c}}_d = \max(\mathbf{c}_d, \tilde{\mathbf{c}}_d)$, with

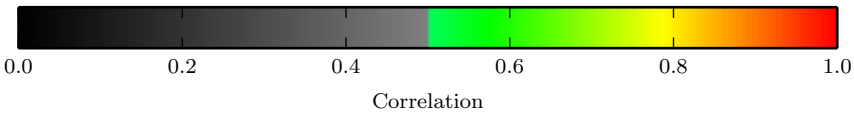


Figure 2-9: Correlation colourmap used in figures 2-10 and 2-11

the lower limit found using:

$$\tilde{\mathbf{c}}_d = \exp\left(-\frac{\max(\|\mathbf{u}\| - 1, 0)^2}{\sigma_l}\right) \tag{2.41}$$

Combined Correlation

The two correlation fields are then combined using element-wise multiplication to find $\mathbf{C}$:

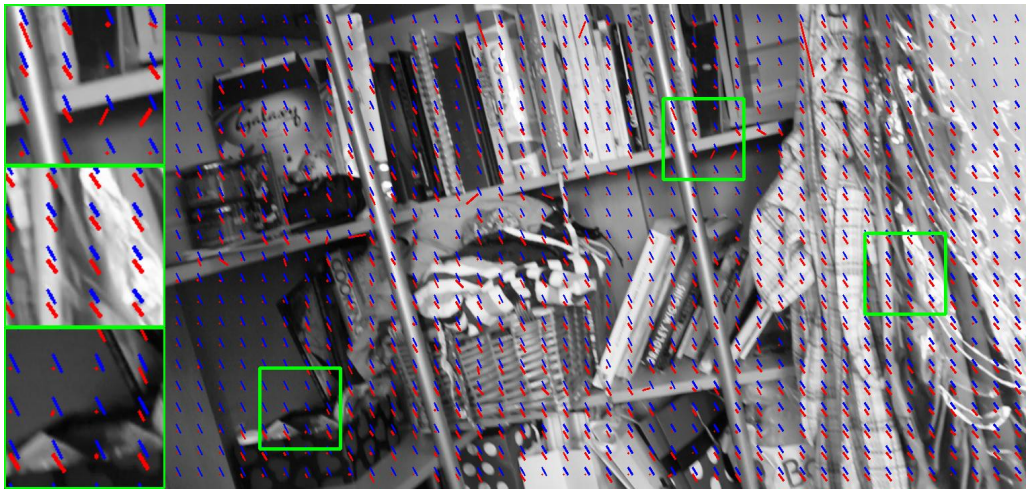
$$\mathbf{C} = \hat{\mathbf{c}}_d \odot \hat{\mathbf{c}}_m \tag{2.42}$$

Figure 2-10 outlines the motion correlation for a frame from a video (see figure 2-9 for details of the colours used in the diagrams). The vectors for both sensor and image motion are visualised in (a). The vectors are generally very similar, but there are some notable exceptions, caused by noise and lack of texture in homogeneous image regions. This is reflected in (b), with $\hat{\mathbf{c}}_d \approx 1$ for regions with low optical flow magnitude.

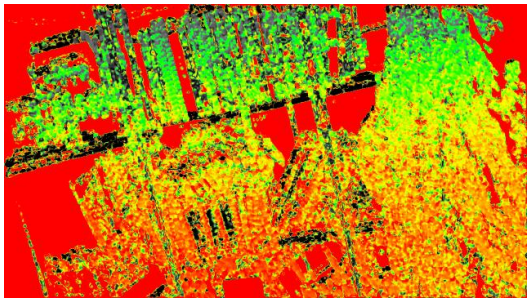
Parallel Implementation

Modern GPUs are massively parallel processors with many hundreds of cores. Much of the content of this thesis is suited to implementation on a GPU, and the performance gains of using a GPU were exploited where reasonably possible. Implementing algorithms on a GPU is not generally straightforward as there are many aspects of the architecture one must be aware of in order to fully exploit it. Indeed, a naive GPU implementation can actually be many times slower than its serial CPU counterpart.

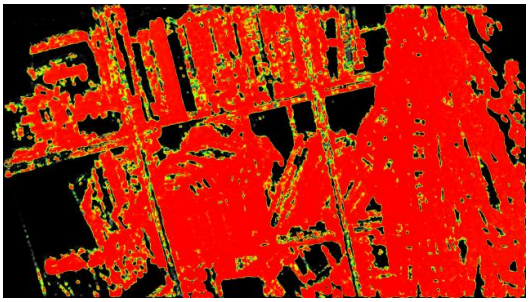
Nvidia's CUDA architecture centres around the concept of a small block of code called a 'kernel' that is executed concurrently by each of the GPU's cores. The division of work is structured around a grid of blocks, with each block containing many threads. The dimensions of the grids and blocks are left to the user's imagination



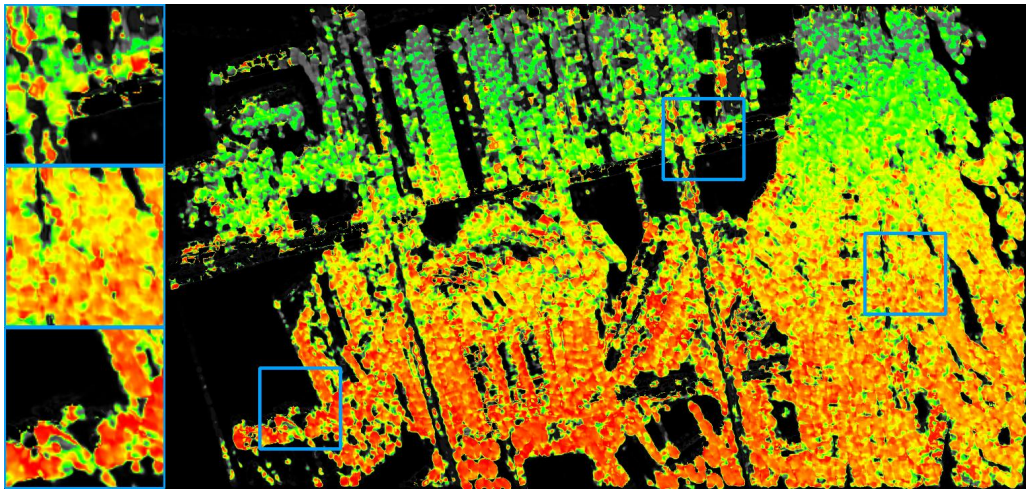
(a) Optical (red) and Sensor (blue) Flow



(b) Direction Correlation $\hat{\mathbf{c}}_d$

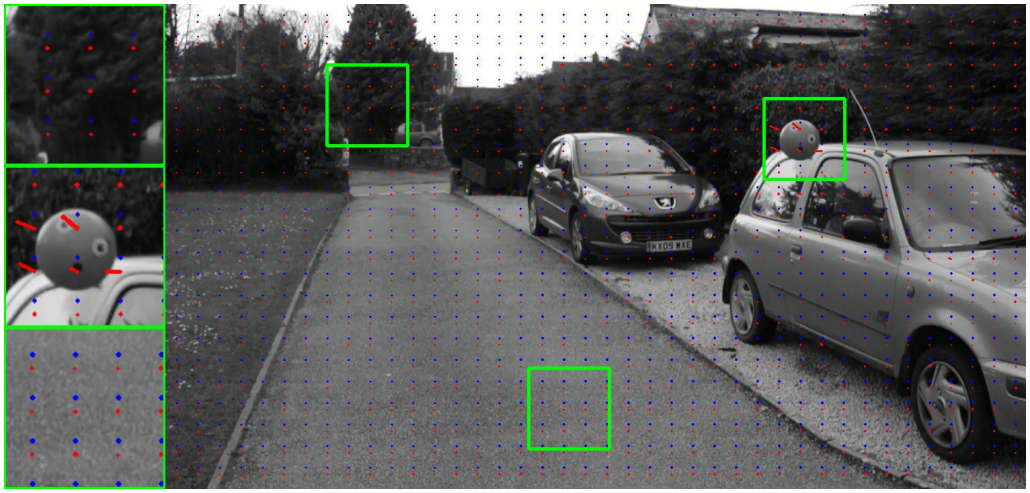


(c) Magnitude Correlation $\hat{\mathbf{c}}_m$

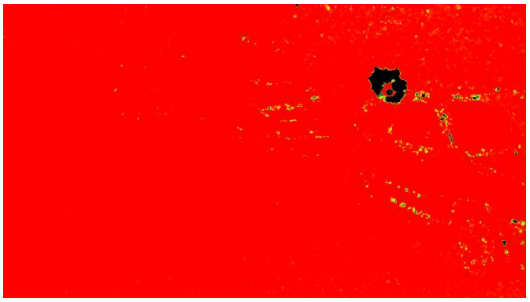


(d) Combined Motion Correlation

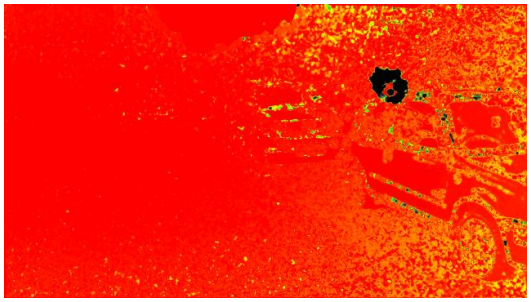
Figure 2-10: Motion correlation between sensors and optical flow. The cut-outs on the left of (a) and (d) are zoomed in regions of the larger image. The top cut-out shows an area with optical flow errors, the middle shows a region with good correlation and the bottom shows a homogeneous region in which the optical flow magnitude is low. See figure 2-9 for colourmap details



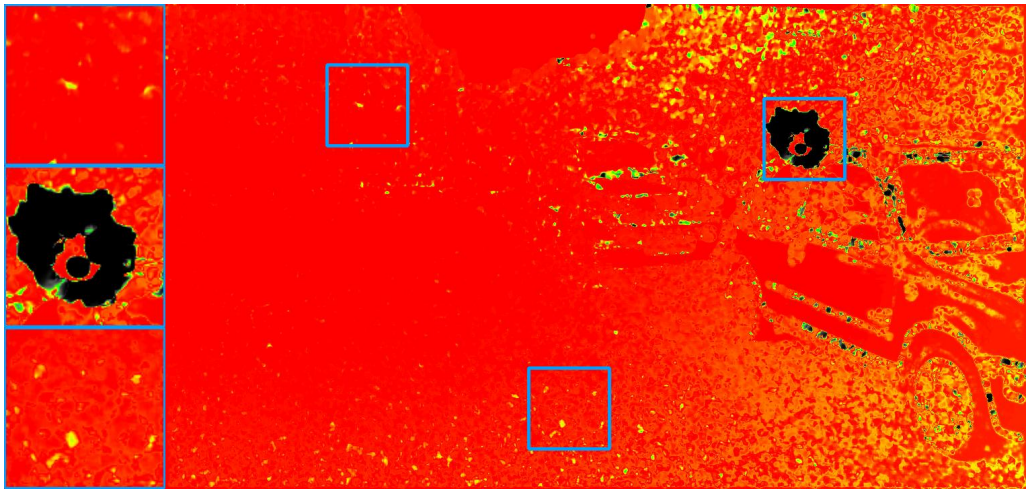
(a) Optical (red) and Sensor (blue) Flow



(b) Direction Correlation $\hat{\mathbf{c}}_d$



(c) Magnitude Correlation $\hat{\mathbf{c}}_m$



(d) Combined Motion Correlation

Figure 2-11: The camera is stationary, so both the optical flow and sensor motion magnitudes are low, leading to a strong correlation across the image with the exception of the region around the ball which is moving. Here, we observe that the low correlation extends beyond the boundary of the ball as a result of the smoothing employed by the optical flow method used. See figure 2-9 for colourmap details

(hardware restrictions notwithstanding). For an image of size 1280 x 720 pixels, it can make sense to use a grid of 80 x 45 blocks, with each block of size 16 x 16. This means that the kernel will be called once for each pixel in the image. If the image is not a convenient size (*e.g.* a multiple of a power of two), it can be helpful to pad the image to such a size, as generally it is less expensive to run the kernel more times than it is to perform bounds checking within the kernel. Whilst it is not always appropriate to divide the work in such a way, we found that this structure of one thread per pixel applied in most cases.

Not all problems are worth solving on a GPU. The overheads of transferring data to and from the device can be relatively high, so it is only worth solving problems of sufficient complexity. One of the most important considerations to take into account when developing for a GPU is memory access. Put simply, accessing the large global memory of a graphics card is slow and should be avoided where possible. Unfortunately, algorithms that require access to many pixels, for example averaging the pixels in a window, can end up spending most of the time waiting on memory accesses.

Fortunately CUDA offers ways to speed up this memory access. Coalesced memory operations see threads simultaneously accessing sequentially addressed linear global memory. The threads grouped in each block have access to a small amount of shared memory that can be used to cache data, with each thread populating a small proportion of the memory which is ultimately then used by all threads. A good example of this would be an averaging filter. Each thread would buffer some of an image from global memory into the shared memory, then find the average of a window centred on the corresponding pixel from the shared memory.

The computation of $\mathbf{C}$ is very well suited to a GPU implementation. We need only transfer the optical flow field $\boldsymbol{\mu}$, homography $\mathbf{H}$ (used to calculate $\mathbf{s}$) and correlation parameters. The result for each pixel in $\mathbf{C}$ requires just one global memory read of $\boldsymbol{\mu}$ and a write for the result.

2.4 Hardware System Overview

The camera motion is captured using our inertial sensor system. We designed a printed circuit board (see figure 2-12) with footprints for inertial sensors (a gyroscope, accelerometer and magnetometer), a Micro-SD card slot and a micro-

controller to connect everything together. We used combined 3-axis sensors in a single package and so we are able to neglect any potential error due non-orthogonal axes, and assume all sensors to be aligned on the same plane.

It was necessary to overcome several difficulties in order to be able to concurrently read the inertial sensors, update an estimate of orientation, synchronise with the camera and write the inertial data and orientation to the memory card, all at a rate of 1kHz. The total average time for each these tasks was greater than the 1ms period but fortunately, with the exception of the sensor fusion algorithm, the operations mostly consisted of reading / writing to the microcontroller's various peripherals, and these could be performed by dedicated hardware in the background, leaving the CPU mostly free.

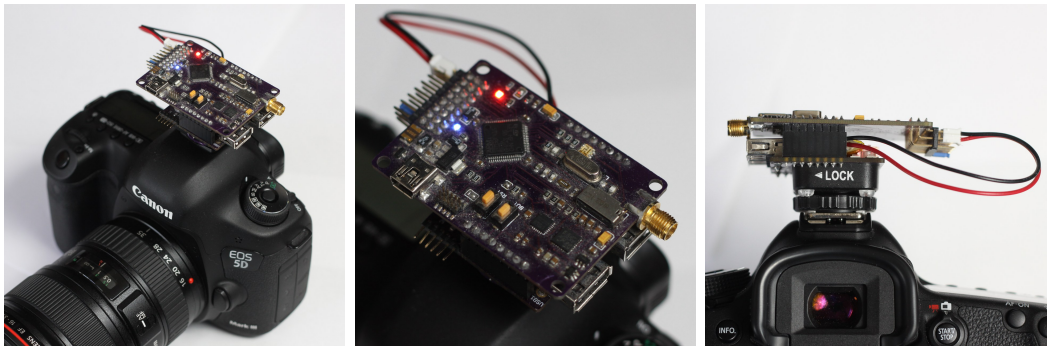
We opted to use ChibiOS/RT[1], which is an embedded real-time operating system. This allowed us to create separate threads for each component; reading the sensors, updating the orientation estimate, writing to flash memory, and a process that coordinates the entire operation, detects when an exposure starts or finishes and generates synchronisation signals.

The system is based around a series of chained producers and consumers, with the data propagating between threads; whenever a sensor is read, the data is placed in a queue, which is then retrieved by the orientation estimator, which subsequently adds its results to another queue and so on. The memory for these queues is statically allocated, with larger queues used when there exists the possibility of delay in using the data, for example writing the data to disk.

2.4.1 Real-Time sensor fusion

We continuously update an estimate of orientation within the embedded system, so that whenever recording starts, there is already an initial estimate of orientation with respect to gravity available. We achieve this by using the optimised method of Madgwick *et al.* [8], and found that the update step is performed comfortably within the 1ms period. Their process updates a quaternion representation of orientation using accelerometer and gyroscope data and performs an optimised gradient descent algorithm to compute gyroscope measurement error. The gain parameter for this algorithm is set such that it prioritises the accuracy of the orientation estimate at the

[1] http://chibios.org



(a) Camera with device (b) Top View (c) Hotshoe mount

Figure 2-12: Motion sensing hardware as mounted on camera

expense of being able to respond quickly to fast rotations, resulting in a smoothed response.

2.4.2 Recorded Data

Each sequence of inertial data is written to a separate file. At the start of the sequence we record the current orientation estimate as a 4 x 32bit floating point quaternion. The biases and scale factors for each sensor are also written in the header as 4 byte floats. These values enable conversion from the raw 16 bit signed values read from the sensors to the offset-corrected readings in θs^{-1} or ms^{-2}. Each data sample is then stored in its raw 16 bit form, direct from the sensor, along with the time in milliseconds, as an unsigned 32bit integer, starting from 0 for the first sample.

Due to the block-based nature of flash storage, writing to a MicroSD card must be done in blocks of 512 bytes, which requires buffering several data packets before writing them all at once. Occasionally the time taken would be much larger, due to file system overheads, so we found we needed a large queue of samples being passed to the writing process, allowing for delays up to 250ms.

2.4.3 Synchronisation for images

The device attaches to the camera via its hotshoe port, which provides a rigid mount. This attachment is shown in figure 2-12. The hot shoe is used to detect when an exposure starts and stops via the flash trigger signal, which is driven low

shortly before an exposure starts and remains that way until it finishes. The signal is connected to an input on the micro-controller and polled every cycle. The inertial data is recorded whilst the signal is low.

2.5 Synchronisation for videos

Detecting the start and stop of a video sequence is more difficult than with images, as the Canon DSLR used for this project has no documented external synchronisation capabilities. We experimented triggering both video and inertial capture with the same external remote, but found its behaviour to be unreliable, with a variable delay between the trigger and the first frame being captured. We worked around this problem by manually starting the inertial capture before the video and then stopping it once the video had finished recording, and then finding the delay between start of the sensor data and images as part of the import process.

The intended use of the inertial sensor and video data was to analyse scenes with moving content, so we required a method of alignment robust enough to cope with an unknown scene with varying depth containing moving objects.

Aron *et al.* [9] correct for a time-varying synchronisation error (rather than searching for the global alignment) between inertial sensors and static-scene image-data by searching in a window around sparse features and using intensity-based optical flow to find the best small shift in alignment at each frame. Hwangbo *et al.* [10, 11] tackled the problem by searching for a known oscillating camera motion at the start of the video sequence, and looked for such a motion in the sensor data.

Karpenko *et al.* [12] attempt to find the alignment in a more general way. They find sparse SIFT[13] feature correspondences between adjacent frames and filter using RANSAC to discard outliers. Coordinate descent is used to find a locally optimal alignment from the given starting position. They repeat the process several times with different starting offsets to increase the chances of finding the global minimum, but this is not guaranteed.

2.5.1 Globally Optimum Alignment of Moving Images with Inertial Sensor Data

The previous discussions on accelerometer data in section 2.2.3 have highlighted difficulties with tracking velocity. For the task of finding the offset between sensor data and imagery, we make the assumption that the most significant camera motion is rotation, allowing us to find the alignment using only the gyroscope data.

The sensors begin capturing gyroscope data at time $t = 0$ at a rate of 1kHz. Shortly after, the camera starts recording with the first frame being exposed after an unknown delay of ΔT seconds. The camera stops recording after having captured N frames, after which the sensor recording is stopped. The goal is to find the offset ΔT between starting the sensor recording and starting video capture.

This amounts to maximising the following probability:

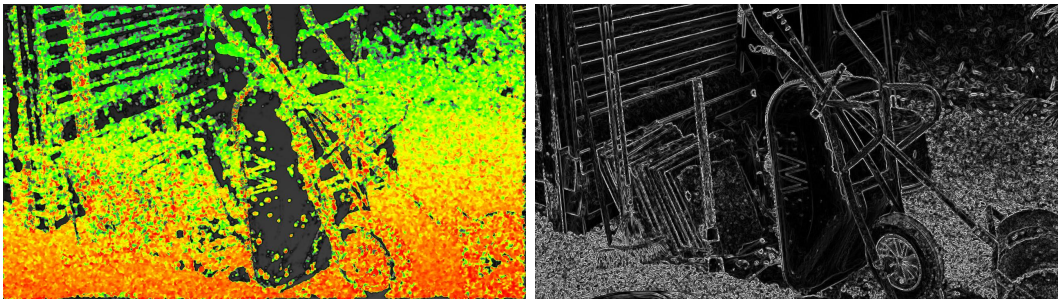
$$P(\Delta T|\mathbf{u}) = \frac{P(\mathbf{u}|\Delta T)P(\Delta T)}{P(\mathbf{u})} \tag{2.43}$$

where $P(\mathbf{u}|t)$ is equivalent to the correlation $\mathbf{C}$ described in section 2.3.3, $P(\Delta T) = 1$ as we assume the offset to be independently distributed. We model $P(\mathbf{u})$ as the confidence in $\mathbf{u}$ which, as previously discussed, we don't expect to be reliable for all pixels. To avoid penalising frames with large homogeneous regions, we experimented with a weighting function $\mathbf{w} \in \mathbb{R}^N$ for each image that acts as a rough optical flow confidence. It was observed that the optical flow field was most reliable around regions with strong texture, so we used gradient magnitude and the Canny edge detector [14] as a simple prior on optical flow reliability. An example of using gradient magnitude as an optical flow confidence is shown in figure 2-13.

We search for the offset ΔT that maximises equation 2.44, corresponding to the strongest correlation over a set $\mathbf{F} \in \mathbb{R}^M$ of all frames indices (in our experiments we found using $M = 20$ frames evenly spaced throughout the sequence to work well):

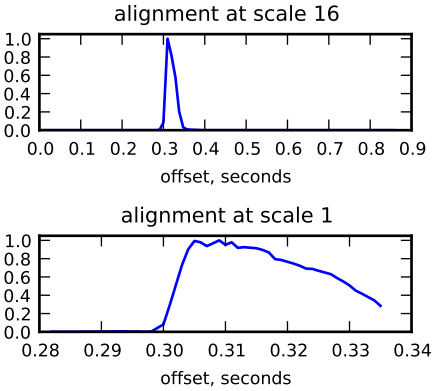
$$\arg\max_{\Delta T} \prod_{i=1}^{M} \sum \frac{\mathbf{w}_i \odot C(F_i, \Delta T)}{\sum \mathbf{w}_i} \tag{2.44}$$

where $C(F_i, \Delta T)$ is the correlation (as described in equation 2.42) of the optical flow computed from two images $\mathbf{I}_{F_i}, \mathbf{I}_{F_{i+1}} \in \mathbb{R}^N$, with the estimated motion between these frames from the inertial sensor data after having been offset by ΔT seconds.



(a) Correlation (b) Gradient Magnitude

Figure 2-13: A simple measure of optical flow confidence using gradient magnitude. Regions with very low gradient magnitude generally have very low confidence



Scene Type	Mean	Std. dev.
Canny Weighting		
Static	7.5ms	3.9ms
Moving	5.3ms	6.3ms
Gradient Weighting		
Static	7.1ms	3.7ms
Moving	6.2ms	6.1ms
No Weighting		
Static	6.4ms	7.5ms
Moving	6.7ms	6.4ms

Figure 2-14: Left: sensor and video data alignment at coarse and fine scales. Right: comparison with ground truth for **10** static scenes and **26** moving scenes

We need only search for ΔT over the range $[0, \tau]$ where τ is the total sensor recording duration minus the video length in seconds. To speed up this process, we start with a coarse spacing of offsets over the entire range, find the optimal alignment and then repeat the process with a finer scale over a smaller range (figure 2-14), centred at the estimated alignment from the previous scale. This approach is well suited to a GPU implementation; we transfer a flow field $\mathbf{v}_i$ and weighting $\mathbf{w}_i$ to the GPU and compute the correlation for each offset before moving onto the next frame, as shown in algorithm 1, avoiding repeated transfers between host and device. In this way it takes less than 2 seconds to evaluate the correlation of 144 different offsets for one 1280x720 image on an Nvidia GTX 560.

Input: Camera orientations $\mathbf{R} \in \mathbb{R}^{3\text{x}3\text{x}\Pi}$ at Π intervals, frame weights $\mathbf{w} \in \mathbb{R}^{N\text{x}F}$, optical flow fields $\mathbf{u} \in \mathbb{R}^{N\text{x}2\text{x}F}$, calibration $\mathbf{K} \in \mathbb{R}^{3\text{x}3}$, frame range $\mathbf{r} \in \mathbb{R}^{F}$, offset range $\mathbf{O} \in \mathbb{R}^{\Pi}$
Output: optimal alignment ΔT
Initialisation:
$\mathbf{E} \in \mathbb{R}^{\Pi\text{x}F} = 0$
$T(j) =$ video time of frame j
for $f \in \mathbf{r}$ **do**
  Transfer $\mathbf{u}_f, \mathbf{w}_f$ to GPU
  for $o \in \mathbf{O}$ **do**
    $\mathbf{R}_n = \text{OrientationAtTime}(T(f) + o)$
    $\mathbf{R}_t = \text{OrientationAtTime}(T(f+1) + o)$
    $\mathbf{R}_r = \mathbf{R}_t \mathbf{R}_n^T$
    $\mathbf{H} = \mathbf{K}\mathbf{R}_r\mathbf{K}^{-1}$
    $\mathbf{C} = \text{MotionCorrelation}(\mathbf{u}_f, \mathbf{H}) \odot \mathbf{w}_f$
    $E_{f,o} = \frac{\sum \mathbf{C}}{\sum \mathbf{w}_f}$
  end
end
$\Delta T = \underset{o}{\text{argmax}} \prod_{f=1}^{F} E_{f,o}$

Algorithm 1: Motion alignment algorithm

Evaluation

The evaluation was performed by comparing the estimated offset against a ground-truth offset. This reference offset was produced by using the inertial sensor hardware to generate a serial bitstream which encoded the time since recording started. This signal is passed through a simple low-pass filter consisting of a capacitor and a resistor, to ensure the frequency is within the audible band, and then captured along with the video via the microphone input.

The audio timing data is encoded using the Linear Timecode protocol (LTC), which we generate at a higher frequency of 100hz, compared to the LTC standard of 30Hz, giving greater time resolution. The audio channel can then be stripped from the video and analysed to find the approximate offset between sensor data and video. This method gives an offset that is accurate to within $10ms$, which corresponds to more than half a frame at 60fps. Whilst this is only approximate, it is sufficient for verifying the general accuracy and reliability of an alignment method over several sequences.

An average alignment error of approximately $6ms$ (as seen in Figures 2-14 and

2-17) was observed in our data. This is a combination of the relatively low time-resolution of the LTC offset, intrinsic delays in the inertial sensing and capturing system, and an unknown offset between first audio samples and exposure of the first frame.

The table in figure 2-14 shows the ground truth evaluation of our alignment method. We see that the alignment of static scenes is distributed more tightly around the mean than for dynamic scenes, yet the mean offset is very similar, demonstrating the robustness of our method.

We compared the performance of the Canny edge detector [14] and discrete gradient magnitude with no weighting and found the weighting functions produced more tightly clustered results (see table in figure 2-14). Using the gradient magnitude as a weighting function is a good choice for this process as it is very cheap to compute, especially on a GPU.

Comparisons against prior work We evaluated our method against the work of Karpenko *et al.* [12], as the most relevant and recent method in the literature. We used their published source code which contained an implementation of their objective function and their gradient descent optimisation step. In order to run this we had to export, amongst other things, sets of sparse feature correspondences between pairs of images from our dataset. We followed their approach of finding SIFT [13] correspondences and then used RANSAC to remove outliers.

Our evaluation covers two aspects: firstly, we used their method to find the alignment using our data, and secondly we also evaluated their objective function over the entire range of possible alignments. The intention is to show that their poor performance in some sequences is as a result of two factors: sparse features leading to a relatively flat objective function and a non-global optimisation approach.

Results

Figures 2-15 and 2-16 show the objective functions (ours is maximised, whilst Karpenko's is minimised), alignment offset estimations and selection of frames from the sequence.

Figure 2-17 compares our method with Karpenko's by showing the proportion of sequences with an offset error (when compared to the LTC offset) below a given margin. Here the benefit of the proposed method is most apparent, with the majority

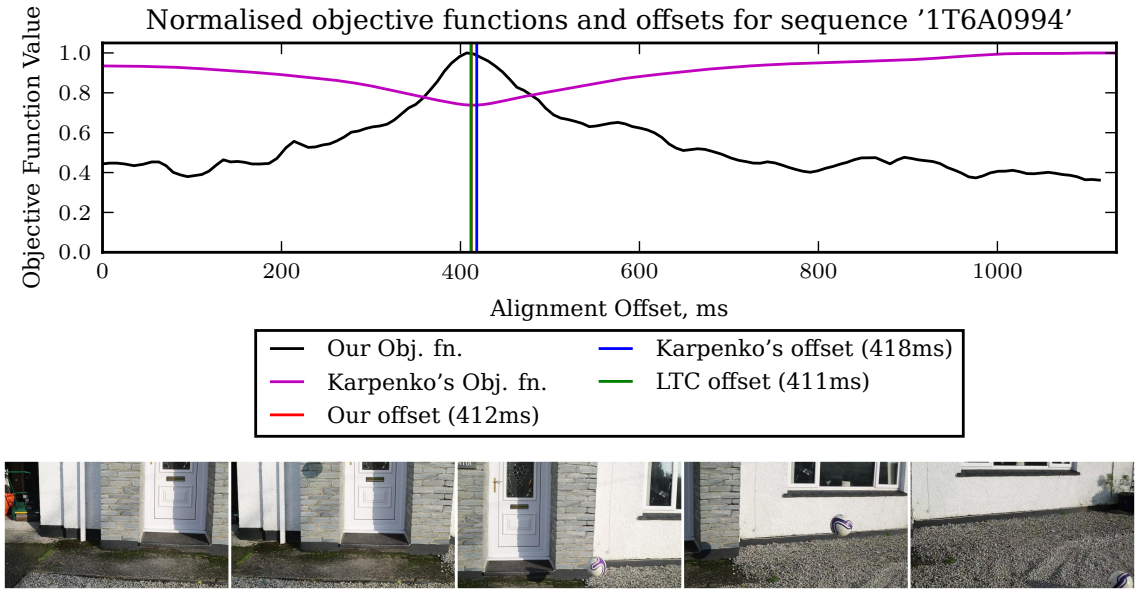


Figure 2-15: This sequence features significant camera motion, whilst the scene is largely static. Karpenko's method performs similarly to ours, with sufficient features to ensure significant variation of their objective function over the range of offsets to find the global minimum.

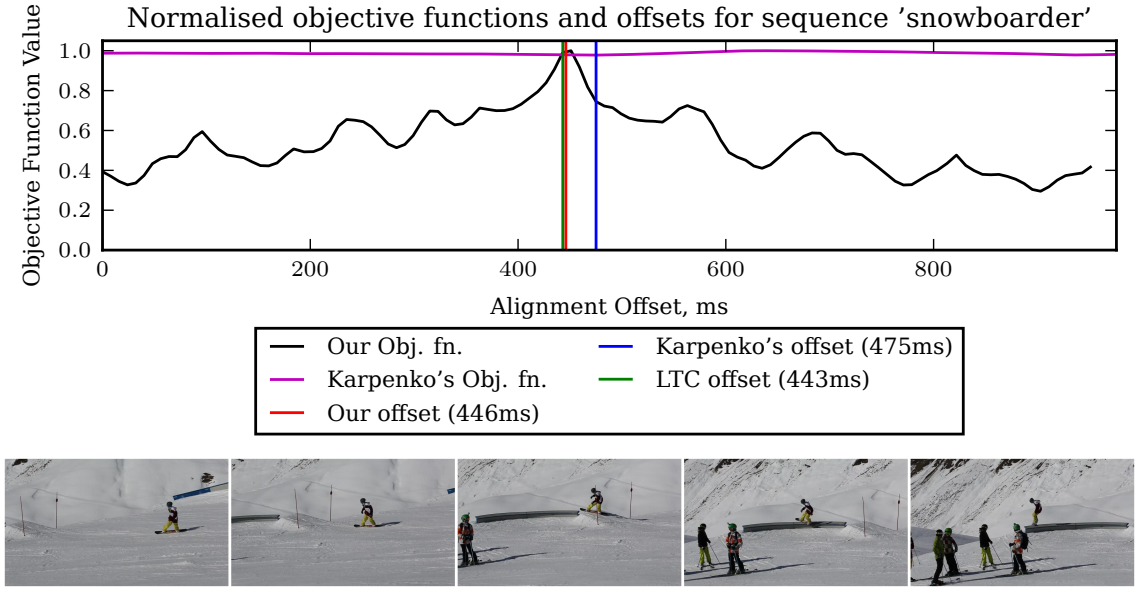


Figure 2-16: In this scene Karpenko's method performs particularly badly, as a result of few distinct, static features. This is reflected in the objective function which does not vary significantly over the offset range, and leads to an inaccurate solution.

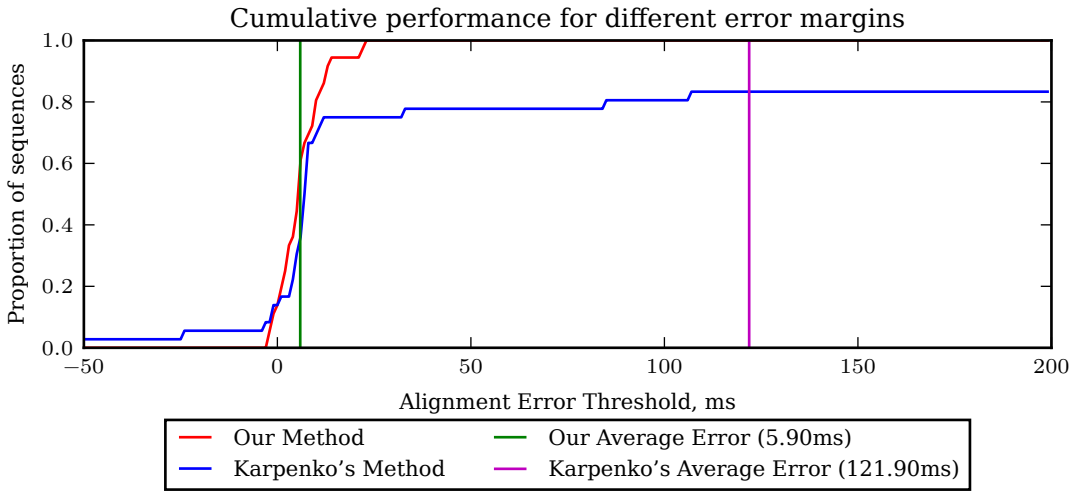


Figure 2-17: Comparison of cumulative error margins. The plot shows the proportion of sequences with the difference between estimated offset and ground truth LTC offset below a particular threshold

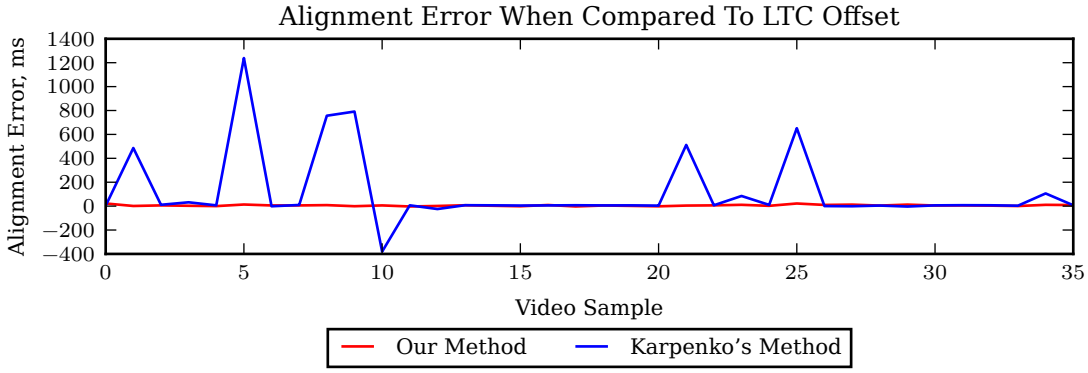


Figure 2-18: LTC Offset Errors for Individual Sequences. This plot demonstrates the consistency of our dense alignment method compared to Karpenko's, where approximately 20% of sequences have an error greater than 1 frame duration

of sequences having an offset error less than 10ms. By contrast, Karpenko's sparse method leads to a poor solution in around 20% of cases.

The reliability and robustness of our method is evident in figure 2-18, with very consistent performance across all sequences. This contrasts with Karpenko's method where several of the sequences have significant offset errors.

Conclusion

We have produced a method for finding the globally optimal alignment of inertial sensor data and video. Our method requires no pre-defined motions and is not dependent on having a scene with distinct content. The results have shown that the proposed dense alignment method performs consistently and achieves a low average error. The primary reason for this is that our simple motion correlation scheme allows for rapid evaluation at a minimal cost and, when combined with a GPU implementation, makes finding the global minimum readily achievable. In addition, the use of dense motion evaluation is beneficial when compared to sparse features in scenes with few prominent structures, particularly if many of these correspond to moving objects, as in the 'snowboarder' sequence. Our dense scheme helps to ensure that there is a sufficient basis upon which to find an optimal solution.

Chapter 3

Optical Flow Confidence

Optical flow is a very useful measure of motion in an image, revealing the dynamics of the scene, and has diverse applications from autonomously-driving cars to video compression. It describes the 'distribution of apparent velocities of movement of brightness patterns in an image'[6], and for a pair of images it is the 2D motion field on the image sensor. This motion is assumed to be the result of a combination of either scene or observer movement.

Whilst the human visual system is very good at understanding the motion of the world, optical flow algorithms generally have failure modes that cause their results to be unreliable, as we've already mentioned in Chapter 2, where we observed that the optical flow result cannot be trusted in all regions of an image. Computing optical flow is a computationally intensive task, and achieving fast results generally comes at the expense of reduced quality.

The reliability of optical flow information is vital if we're to analyse the motion, and when accuracy is not prevalent it is desirable to know where the flow can and can't be trusted. In the previous chapter we experimented with some crude approximations for an optical flow confidence measure using gradient magnitude. Here we discuss in more detail the causes of erroneous optical flow results and present a novel method for learning an optical flow confidence from real image data and inertial sensor readings, rather than the synthetically-rendered or meticulously-hand-labelled data that has been prevalent up until now. The main benefit of this approach over one using synthetic data is the ability to easily capture and target training data. We additionally show an improved performance when using real data and where the data closely matches the target environment.

In this work we are not concerned with the evaluation of different optical flow algorithms, and use the work of Farnback *et al.* [7] with the parameters set to produce a fast result. Our main motivation was to investigate whether or not inertial sensor data can be used as a basis for learning an optical flow confidence measure, with a preference towards simple and efficient methods.

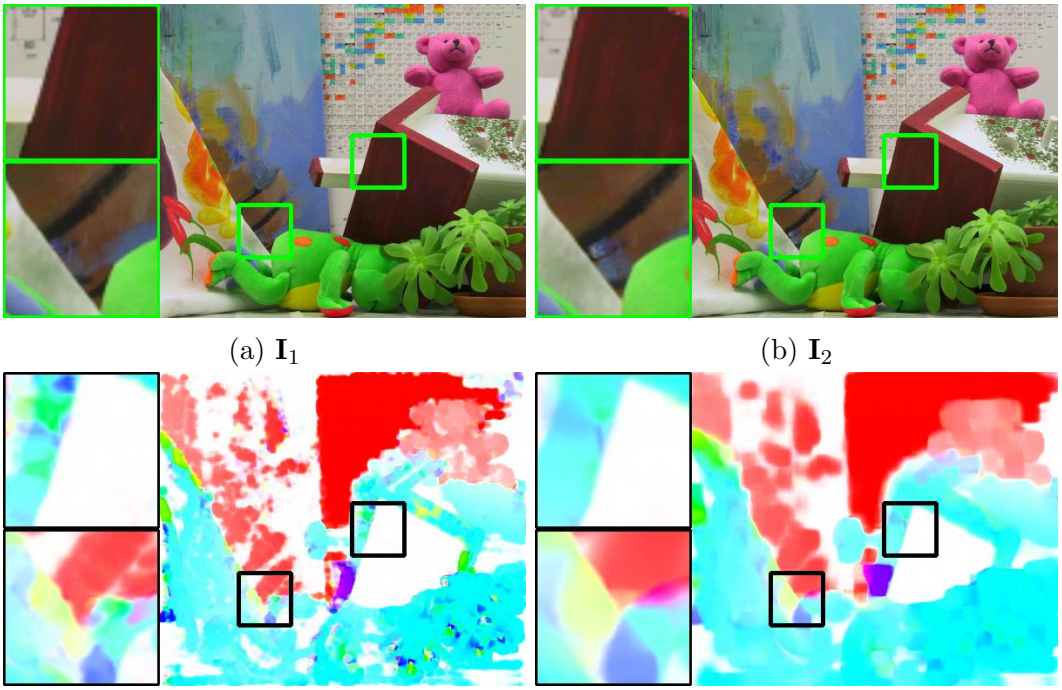
There are advantages associated with using real images and cheaply captured inertial data over synthetic or hand labelled datasets. In the case of the former, Meister *et al.* [15] found the flow in synthetically generated images to be 'too smooth', with different spatial distributions of errors between real and synthetic data. The key disadvantage associated with hand-labelled datasets is the amount of work required in their production. By contrast, the ability to quickly film any scene without intensive manual labour is desirable, allowing for experimentation with different scene types or visual phenomenon with minimal effort. Such functionality could also be extended to real-time applications, to enable an optical flow confidence measure that continuously adapts to its environment.

We define the optical flow confidence $\boldsymbol{\phi} \in \mathbb{R}^N$ in the range $[0, 1]$, as the probability that the optical flow vectors in $\mathbf{u} \in \mathbb{R}^{2N}$ are a true representation of motion for each pixel. This is a measure of how much faith we have in $\mathbf{u}$.

One option for producing $\boldsymbol{\phi}$ is to pose the problem as one of finding the error value for each pixel. This is difficult as the errors are not correlated linearly and can be drawn from a random distribution in the case of noise or exhibit systematic bias where the optical flow model assumptions prefer a particular type of flow field. A more achievable alternative is to compute the optical flow confidence as the probability that the optical flow has an error within some upper bound, with the bound varying depending on the application; this is the approach we use.

3.1 Optical Flow

We've already given a brief overview of optical flow in Chapter 2, and note that a comprehensive review of this extensive field is beyond the scope of this thesis. Instead, we concentrate on putting the theory of optical flow into the context of our work, and describe some common facets of optical flow methods. Without going into algorithm-specific detail, we relate these to sources of optical flow error. For completeness, we state that the optical flow vectors $\mathbf{u} \in \mathbb{R}^{2N}$ describe the motion



(a) $\mathbf{I}_1$ (b) $\mathbf{I}_2$

(c) Optical Flow - 5 pixel window (d) Optical Flow - 15 pixel window

Figure 3-1: Two frames of 'Teddy', from the Middlebury dataset[16] shown in (a) and (b). The cutouts show the same region in each image. The optical flow fields are colour-coded according to figure 3-2, with (c) showing the optical flow result with a small smoothing window, and (d) the result with a larger smoothing window

between two images, $\mathbf{I}_1, \mathbf{I}_2 \in \mathbb{R}^N$.

An example of optical flow can be seen in figure 3-1, with the two source frames shown as well as flow fields resulting from different algorithm parameters. The flow fields are visualised using the popular colour coding of [16], where the colour represents flow direction and the saturation describes the flow magnitude up to a maximum of 10 pixels, with white being a stationary flow. The full spectrum of the flow notation is shown in figure 3-2.

3.1.1 Optical Flow Estimation

The majority of optical flow algorithms solve the problem of estimating optical flow by minimising a global objective function comprised of two weighted terms:

$$\mathbf{E}(\mathbf{u}) = \mathbf{E}_{\text{data}}(\mathbf{u}) + \lambda \mathbf{E}_{\text{prior}}(\mathbf{u}) \tag{3.1}$$

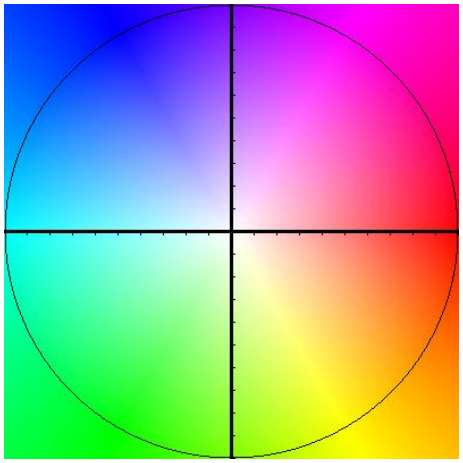


Figure 3-2: Optical flow colour coding as in [16]. The saturation represents flow magnitude up to a maximum of 10 pixels, the colour indicates the direction

The *data term* models how well a flow field represents the pixel-wise motion between $\mathbf{I}_1$ and $\mathbf{I}_2$, and is based on the assumption that some visible image quantity is conserved between the frames. Commonly, *brightness consistency* assumes pixels to retain the same colour and intensity as they flow between the images, and so prioritises a flow that minimises variation in appearance. Other features of the data term, such as illumination and blur, are reviewed by Baker *et al.* [16]. The data term results in a per-pixel penalty that is then aggregated over the entire image. Finding the motion that minimises the energy of the data term is an ill-posed problem as there can be many different flows that result in a pixel of similar appearance.

The *prior term* applies a constraint on the properties of the flow field. Typically this is used to enforce local smoothness, as it is reasonable to assume that the motion of one pixel is likely to be strongly correlated with that of its neighbours. A simple form for $\mathbf{E}_{\text{prior}}$ is one that minimises the first-order gradients in $\mathbf{u}$, leading to a smoothed flow field. Alternative forms are discussed by Baker *et al.* [16].

3.1.2 Sources of optical flow errors

A confidence measure expresses the uncertainty associated with a flow estimate, and so an understanding of the causes of errors in flow estimation is necessary. We now discuss some causes of optical flow error, referring to examples that demonstrate the effects.

Non-unique solutions

In homogeneous or repetitive regions there are many motions that yield a similar looking part of the image, and so the energy minimisation function for the data term is likely to find multiple global minima. The optical flow prior term will smooth the result leading to a flow field with no motion, apart from those regions near strong features, where the flow will be influenced by neighbouring motion.

This effect can be witnessed on the roof in the upper cut-out of the images shown in figure 3-1. The inner regions have no flow, due to lack of texture. However, comparing figures 3-1c and 3-1d, we observe that the region with flow extends further into the centre of the roof with the larger smoothing window.

It can be the case that an iterative optimisation method gets caught in a local minimum or saddle point, and fails to recover, or that varying initial conditions yield different solutions. In either case, such failings are not easily predicted, and are specific to the optical flow method.

Occlusion

Occlusion and dis-occlusion are opposite effects and are determined by which image of the pair the optical flow motion is taken relative to. Occlusion occurs as the result of either scene motion or parallax induced by motion of the camera, causing regions that were visible in one image to be hidden (occluded) in the next. When considered in reverse, we have the case of dis-occlusion, in which regions that could not be seen in one image are then visible in the next. Such regions have undefined optical flow, as there are points in one image with no corresponding points in the other image. The flow estimated in these regions will vary depending on the method, but it is typical to see either noise or smoothed flow from surrounding regions. The effect of this is apparent in the lower cut-out of the example in figure 3-1, where we see a lot of variation in the optical flow direction. It can also be observed that variability of flow direction in occluded regions is lower with a larger window size (figure 3-1d).

Scale

It is common for optical flow techniques to handle large scale motion using an image pyramid, in which optical flow is computed on the image at different sizes. The consequence of this is that small scale features can disappear when down-sampled,

reducing the reliability of the result.

Summary

The examples highlight how the chosen optical flow method has an impact on which regions of the flow field are reliable, implying that an optical flow confidence measure needs to be aware of the parameters in order to produce a good result.

3.2 Related Work

Optical flow confidence has been worked on as an internal component of optical flow methods, as a means of evaluating the performance of optical flow methods, and in its own right for assessing the reliability of a flow field. In this section we review the literature relating to optical flow confidence.

Barron *et al.* [17] demonstrate the value of evaluating optical flow fields, and show how performance can vary greatly between different techniques. In [18], Bainbridge-Smith and Lane evaluate several different early confidence schemes. Márquez-Valle *et al.* [19] categorize confidence measures by their accuracy and capabilities for error bound prediction, and describe a framework for assessing the quality.

3.2.1 Analytical Image Intensity

Early works were focused largely on the local statistics of image intensity, and formed measures from an analytical standpoint. Uras *et al.* [20] restrict the estimation of flow to regions where the determinant of the local Hessian is non-zero. Simoncelli *et al.* [21] compute an expression for the probability of a flow vector based on image gradients, and produce a two-dimensional probability distribution, on the assumption that an increase in image contrast (signified by image gradient) leads to an increase in flow certainty. Jahne and Peter [22] produce a confidence measure based on the eigenvalue decomposition of the structure tensor or temporal gradient of the images.

3.2.2 Algorithm Specific

Some confidence measures are designed with specific optical flow algorithms in mind. Bruhn and Weickert [23] assert that the inverse of the local energy function for an

optical flow algorithm is a good fit for variational methods. Kybic *et al.* [24] estimate an error magnitude for each pixel in an image. They use bootstrap re-sampling, in which they repeatedly estimate optical flow for different random subsets of pixels. Their method works for optical flow algorithms that can be posed as a per-pixel minimisation problem.

3.2.3 Learned Models

A problem with hand-tuned or algorithm-specific confidence measures is their lack of adaptability to changes in optical flow method or parameters. Such flexibility can be highly desirable in applications where the optical flow parameters are not known in advance, *e.g.* in real-time situations where the parameters are tweaked according to processing power. Kondermann *et al.* [25] describe a confidence measure that is independent of the optical flow method and learn a probabilistic motion model in local windows from training data.

Aodha *et al.* [26] present a method for learning a confidence measure. They use a Random Forests [27] classifier to learn the relationship between multiple spatio-temporal features and their training data. The feature set comprises of gradient magnitude, distance from edges and object (defined using gPb [28] segmentation) boundaries, Their method is not restricted to a particular class of optical flow algorithm, and they do not make any scene assumptions. Their training criterion is formed by binary thresholding endpoint errors, produced from synthetically rendered image pairs. In their work, they state that the gPb (global probability of boundary [28]) feature is the most important of the features they test in terms of getting good confidence results. This is likely because their dataset is largely formed of objects that move relative to their background as a result of camera parallax, as is typical for many optical flow verification datasets. In this scenario, the gPb segmentation excels at localising the most significant boundaries in an image.

3.3 Learned Optical Flow Confidence from Sensor Data

Our method for producing optical flow confidence overcomes many of the shortcomings of the previous work. We do not make any assumptions about the optical flow

algorithm for which confidence is to be measured, and allow real images to be used as training data.

We use the approach of Aodha *et al.* [26], and pose confidence estimation as a binary supervised learning problem, whose goal is to estimate a confdience value $\boldsymbol{\phi}$ for each pixel in an image. The training set $\mathcal{D}$ then takes the form:

$$\mathcal{D} = \{(\mathbf{x}_i, c) | \mathbf{x}_i \in \mathbb{R}^d, c_i \in \{0, 1\}\}_{i=1}^n \tag{3.2}$$

where d is the dimensionality of the feature vector $\mathbf{x}_i$, $\mathbf{c}$ is the training vector and n represents the number of samples used for training. The training set is a collection of input features $\mathbf{x}_i \in \mathbb{R}^d$ and output labels $c_i \in \{0, 1\}$. The result of the training process is a learned mapping from a set of input features $\mathbf{x}_i$ to the probability that those features correspond to the class of either reliable ($c = 1$) or unreliable ($c = 0$). For our purposes, we're interested in the probability that the optical flow at any pixel is reliable (given some threshold), and so we set $\Phi_i = P(c_i = 1 | \mathbf{x}_i)$.

In Aodha's work, they assign class training labels based on the end-point-error (EPE), which is the magnitude of the difference between an estimated and ground truth flow field. In contrast, we experimented with using our motion correlation (MC) function C as defined in Chapter 2:

$$c_i = \begin{cases} 1 & \text{if } C_i \geq 0.5 \\ 0 & \text{if } C_i < 0.5 \end{cases} \tag{3.3}$$

We will give an evaluation of the performance of training with motion correlation compared to end-point-error in section 3.5.

3.3.1 Classification Method

The Random Forest [27] classifier is an ensemble method that uses 'bagging' (bootstrap agregating [29]) to train trees on different subsets of data, which are selected randomly with replacement. The predictions of the trees are aggregated to produce a robust model. It is a good choice for this classification problem as it is robust to noise, produces accurate results [30], can handle large datasets such as images, and has the advantage of being inherently parallel.

The training labels defined in $\mathbf{c}$ can be quite noisy due to errors in motion estimation from the inertial sensor data and the potential for scene motion. We

found that even when a few frames had significant errors, the training process was robust enough to produce an accurate result.

3.3.2 Features

The feature vector $\mathbf{x}$ is a d dimensional representation of a given image pair $\mathbf{I}_1$ and $\mathbf{I}_2$ with optical flow $\mathbf{u}$, computed for each pixel in $\mathbf{I}_1$. The features encode what we believe to be salient traits that are informative for deciding whether or not optical flow is reliable. These features form the basis on which we train our classifier, and are also used for predicting the confidence for an input image. The consequence of this is that if we wish our optical flow confidence measure to be quick to compute, then it stands that our feature vector must also be quick to produce.

We analysed the features used in [26] and built our feature vector out of those we found most relevant to our work and motivations. Part of our assessment involved looking at the complexity of producing each feature, and deciding whether the cost was worth the improvement in the confidence measure.

In our video sequences, we did not experience as much parallax motion as in the dataset used by Aodha *et al.* [26], and did not find the expensive gPb [28] feature to be necessary.

Gradient Magnitude

Textured regions are readily tracked by optical flow algorithms, as they tend to give strong features that result in unique mappings between frames. Following on from Aodha *et al.* [26], we use gradient magnitude as a measure of texture magnitude. Figure 3-3 shows the result of computing $\mathbf{g}$ on $\mathbf{I}_1$ from figure 3-1a. The upper cut-out region has strong texture and we see in figure 3-1d that the flow is reliably computed in this region.

Gradient magnitude is an attractive feature to use as it is very quick to compute. We implement g as the magnitude of the discrete differences in x and y,

$$g(x, y) = \|\nabla \mathbf{I}_1(x, y)\| \tag{3.4}$$

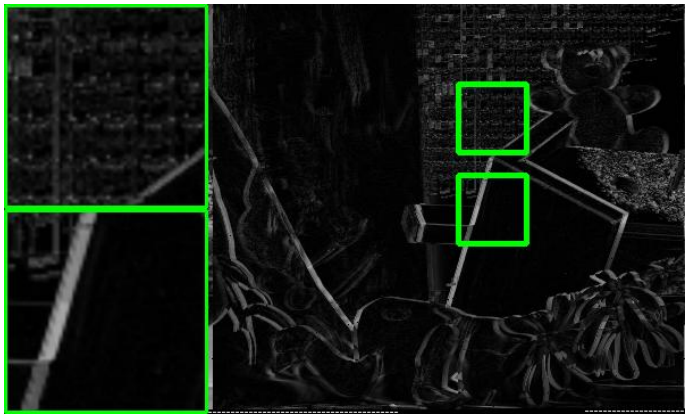


Figure 3-3: Gradient Magnitude $\mathbf{g}$ of $\mathbf{I}_1$ from figure 3-1a

where ∇ is comprised of discrete gradient operators:

$$\partial_x(x,y) = \mathbf{I}(x,y) - \mathbf{I}(x+1,y) \tag{3.5}$$

$$\partial_y(x,y) = \mathbf{I}(x,y) - \mathbf{I}(x,y+1) \tag{3.6}$$

This feature is very simple to implement on a GPU as the result of each pixel is dependent on that of its immediate neighbours, allowing for coalesced memory access across synchronised threads.

Edge Distance

Edges are likely to correspond to a boundary between regions of discontinuous motion, and also generally mark the boundaries of homogenous regions. This characteristic is apparent in figure 3-1, where it is observed that the centre regions of the roof have no flow, whereas the regions towards the edge of the roof, whilst having little texture, end up with motion as a result of the flow field smoothing. This feature measures the distance from each pixel to the nearest strong edge. The 'strong edge' criteria is computed using the canny edge detector [14], and a strong edge is determined as one whose magnitude exceeds the threshold τ_{ed}:

$$d(x,y) = distTrans(||\nabla \mathbf{I}_1(x,y)|| > \tau_{ed}) \tag{3.7}$$

We show the result of computing $\mathbf{d}$ on $\mathbf{I_1}$ from figure 3-1a in figure 3-4, where

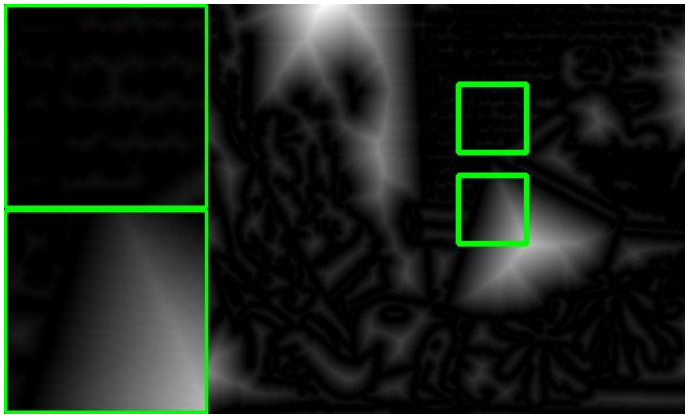


Figure 3-4: Edge Distance **d**, normalised for visualisation purposes

high values are found in regions in the centre of objects such as the roof.

This feature is difficult to optimise for a GPU implementation as the result for each pixel requires checking the content of many surrounding pixels, involving slow global memory accesses. A naive implementation of this algorithm on a GPU would be much slower than the serial CPU version as a result.

Scale

We compute each of our features on images resized by scales 1.0 and 0.5. The result is then upscaled back to the original size before being added to the feature vector, effectively adding an extra feature set for each scale.

Temporal Gradient

We slightly modify Aodha's [26] temporal feature as theirs is computed over the flow field of several optical flow methods. We take the gradient magnitude of **u** in x and y directions separately:

$$t_x(x, y) = \|\mathbf{u}_x\| \tag{3.8}$$

$$t_y(x, y) = \|\mathbf{u}_y\| \tag{3.9}$$

This feature helps identify regions with discontinuous flow motion.

3.3.3 Training Data Selection

We train our confidence measure from a selection of sequences shot by a moving camera as it pans around a variety of static scenes. From the set of videos we randomly select frames and take a random sample of pixels from each. We only train from image regions where the motion magnitude (according to the sensor data) exceeds a threshold. This is because regions with low texture will register strong correlation with sensor data for a static camera, and we do not want this leading to such regions being learned as reliable.

3.3.4 Confidence Estimation

The output confidence estimate ϕ is calculated by computing the feature vector for every pixel in the target image, and then finding the probability of the trained random forests decision tree assigning a class label of 1, which indicates a reliable optical flow result.

3.4 Results

We trained a confidence measure using videos of static scenes without significant depth variation filmed by a moving camera, with the corresponding inertial data used to produce the training criterion. The images in table 3.1 are from Aodha's dataset [26], with the confidence images produced by our confidence measure trained on inertial data. The end-point-error (EPE) is found as the magnitude of the difference between their ground truth flow fields and the estimated optical flow.

A qualitative analysis of the results shows that in general the confidence measure generally produces the expected results, with the confidence being high (white) for regions with low EPE (black). The confidence measure has problems on the edges of objects where occlusion occurs, such as the crates in *009_Crates1*; this error is most apparent in the 'Confident Errors' column of the table, where there is significant error around the object boundaries. This could likely be improved by the use of the gPb feature of Aodha *et al.* [26], but we opt to leave it out due to its computational cost and the reduced parallax in our target sequences.

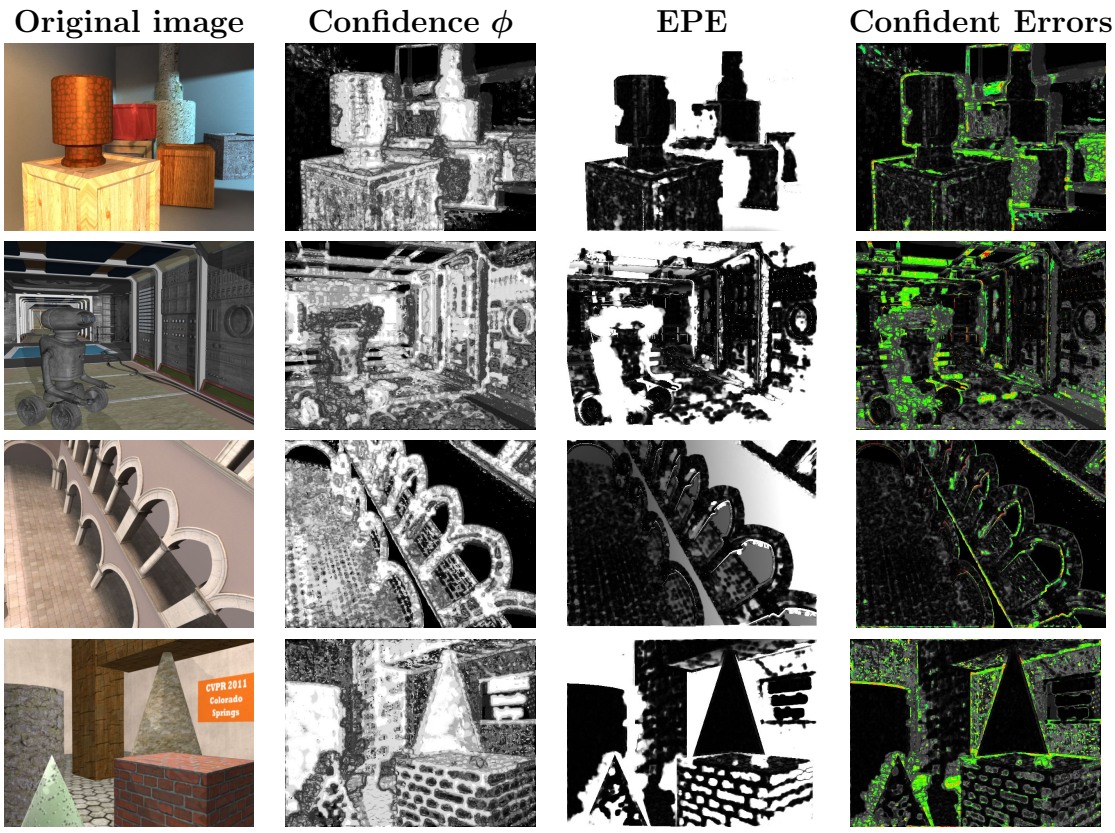


Table 3.1: Optical flow confidence estimation for four samples from the dataset of Aodha *et al.* [26]: *009_Crates1*, *017_Robot*, *019_Sponza2* and *026_Brickbox1t1*. The black and white images range from black (lowest) to white (highest). The second column shows the result of our confidence measure when trained with inertial data. The third column is the end-point-error between the measured optical flow and the ground truth. The fourth column is a heatmap showing the EPE for the portions of the image for which we are confident about the optical flow, calculated as $\phi \min(EPE, 5)$. In other words, 'Confident Errors' shows the areas where we are most wrong to be confident. These images are scaled to show which parts of the image have the most image, rather than as an indication of absolute error.

3.5 Evaluation

The evaluation of the performance of our inertially-trained optical flow confidence measure has two aspects. We begin by evaluating its performance using optical flow ground truth data, to establish the validity of using inertial sensor data as a basis for training an optical flow confidence measure. We then demonstrate the benefit of being able to train using real image data by showing improved performance. The main focus of this evaluation is not the suitability of our feature set when compared to others, but rather to demonstrate that inertial sensor data can be a suitable alternative to synthetic or hand-produced datasets, and that it offers benefits over using synthetic data to train an optical flow confidence measure.

In each case we are comparing a model trained using real images against one trained using synthetic data. We evaluate the performance of an optical flow confidence measure in the same way as Aodha *et al.* [26], and assess the measure's ability to rank the most reliable regions over those that are less reliable.

For each image used for evaluation we have a ground-truth flow field and an estimated optical flow field. These are used to find the end-point-error (EPE) which is the magnitude of the difference between the vectors in each flow field. This magnitude is high where the optical flow has errors and low where the optical flow is accurate. The EPE image is then compared to a confidence image $\boldsymbol{\phi}$ produced by each optical flow confidence measure.

An optimum optical flow confidence measure would yield the highest confidence for the pixel with the lowest EPE, the second highest confidence for the pixel with the second lowest EPE and so on, with the lowest confidence given to the pixel with the highest EPE. The comparison is performed by first sorting the EPE in ascending order. Then, for a given optical flow confidence image, we find the average EPE (aEPE) for *e.g.* the 1% of pixels with the highest confidence and compare this to the aEPE of the 1% of pixels with the lowest EPE from the optimal sorting. This process is repeated across a range of thresholds.

A comparison between optical flow confidence measures can then be made by comparing aEPEs at different thresholds. The better a confidence measure, the lower the aEPE. This is easily interpreted in graph form as shown in Figure 3-9. This graph shows the optimal result as well as the performance achieved when using a simple, naive confidence measure - image gradient.

3.5.1 Training Sets

Our evaluation compares optical flow confidence measures trained using either synthetic data or real images. Here we briefly describe each dataset.

Synthetic Training Data

This training set is built using Aodha's synthetic data. It comprises 24 frame pairs, of which the first frame of each pair is shown in Figure 3-5. A significant feature of this dataset is varied scene depth containing parallax and occlusion, with a mixture of objects of varying shapes and textures.

Real Training Data

Our real training data is comprised of 7 short videos of a variety of static objects. These include detailed objects such as foliage and jewellery, straight edged objects such as shelving units and signs, curved low-contrast objects, high-contrast branches and textured walls. A sample of frames from each of these sequences is shown in Figure 3-6.

3.5.2 Evaluation against Synthetic Ground Truth

We evaluate the performance of an optical flow confidence measure trained on inertial data by using the optical flow ground truth dataset from [16]. Each evaluation sample consists of an image pair and a ground truth optical flow field $\mathbf{u}^* \in \mathbb{R}^{2N}$. An estimate of optical flow $\mathbf{u} \in \mathbb{R}^{2N}$ and confidence measure $\boldsymbol{\phi}$ are then produced from the image pair and first image respectively.

The graphs in figures 3-7 and 3-8 compare several different measures:

- **Inertial Static MC** - Confidence trained on real images of static scenes and inertial data, with motion correlation used as training threshold criteria
- **Inertial Moving MC** - Confidence trained on real images of moving scenes and inertial data, with motion correlation used as training threshold criteria
- **Inertial EPE 0.25** - Confidence trained on real images and inertial data, with $EPE < 0.25$ used as training threshold criteria

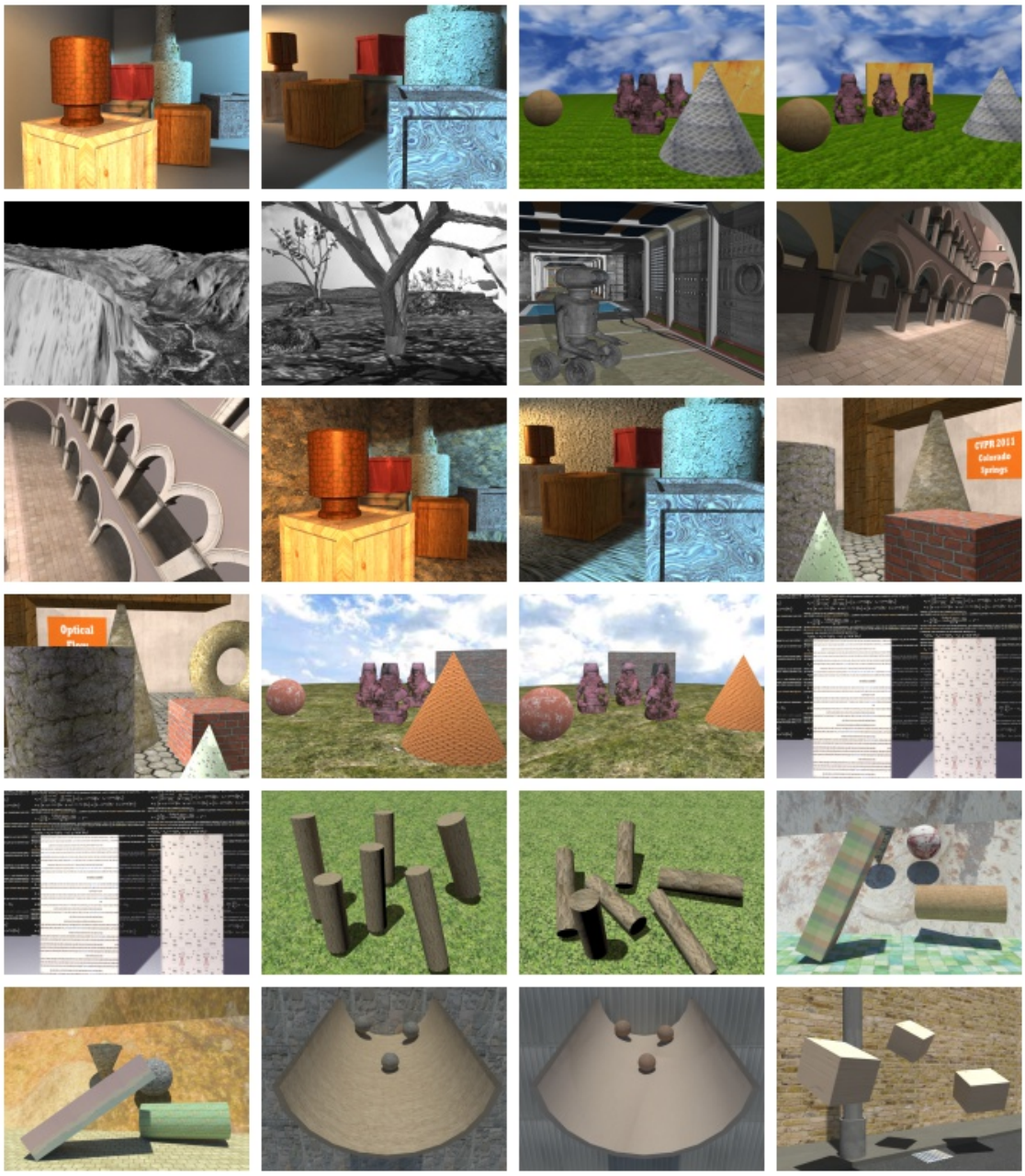


Figure 3-5: Aodha's training dataset. Here we show the first frame of each pair.

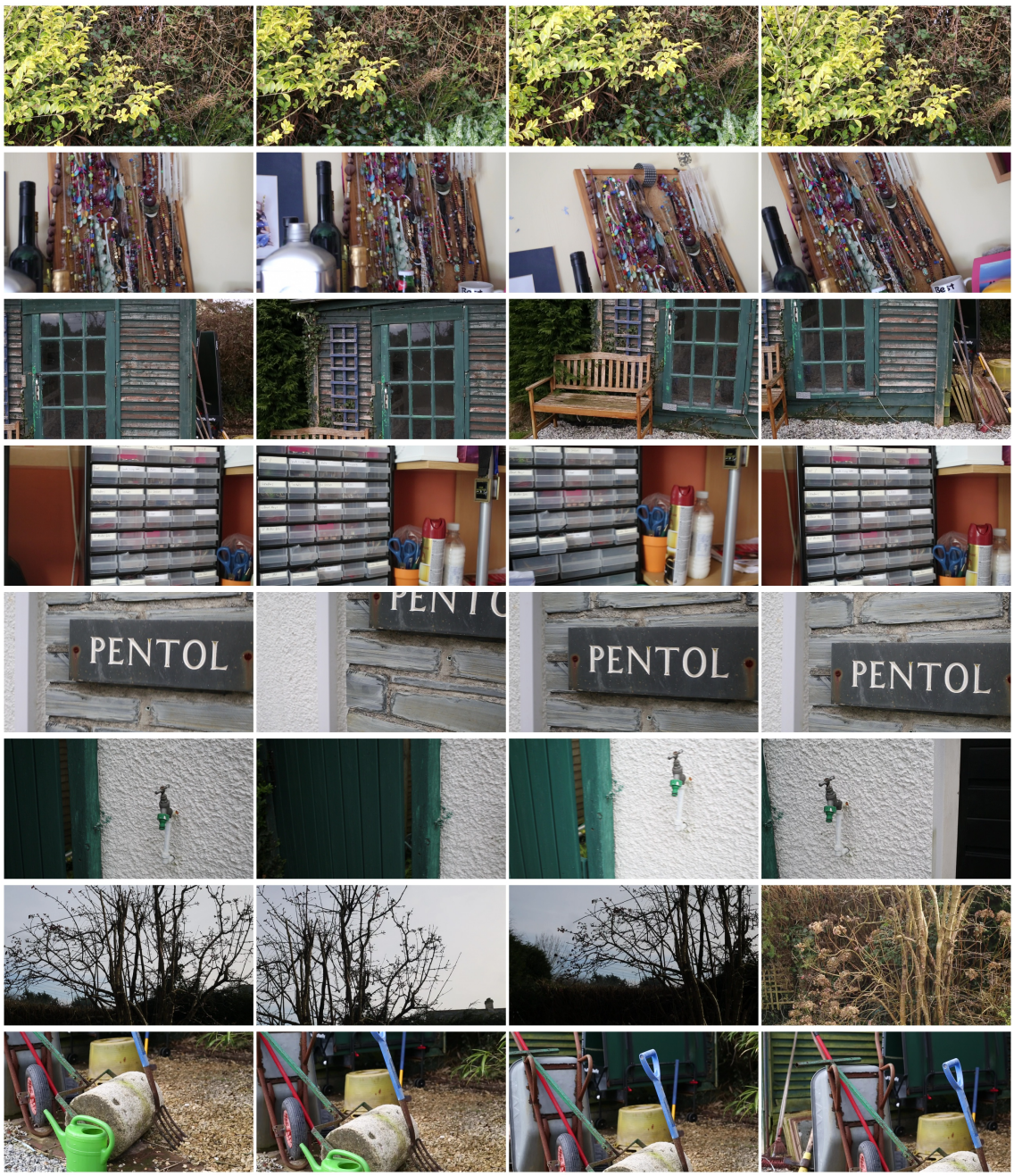


Figure 3-6: Sample frames from static sequences used to produce the real-image evaluation dataset

- **Inertial EPE 2.0** - Confidence trained on real images and inertial data, with $EPE < 2.0$ used as training threshold criteria
- **Synthetic EPE 0.25** - Confidence trained on Aodha *et al.* 's synthetic dataset [26], with $EPE < 0.25$ used as training threshold criteria
- **Gradient Mag** - Gradient magnitude used as a very simple optical flow confidence measure
- **Optimum** - The optimal result, obtained by sorting by EPE

The MC training criteria was calculated with parameters $\sigma_\mu = 100, \kappa = 120, \beta = 0.5$.

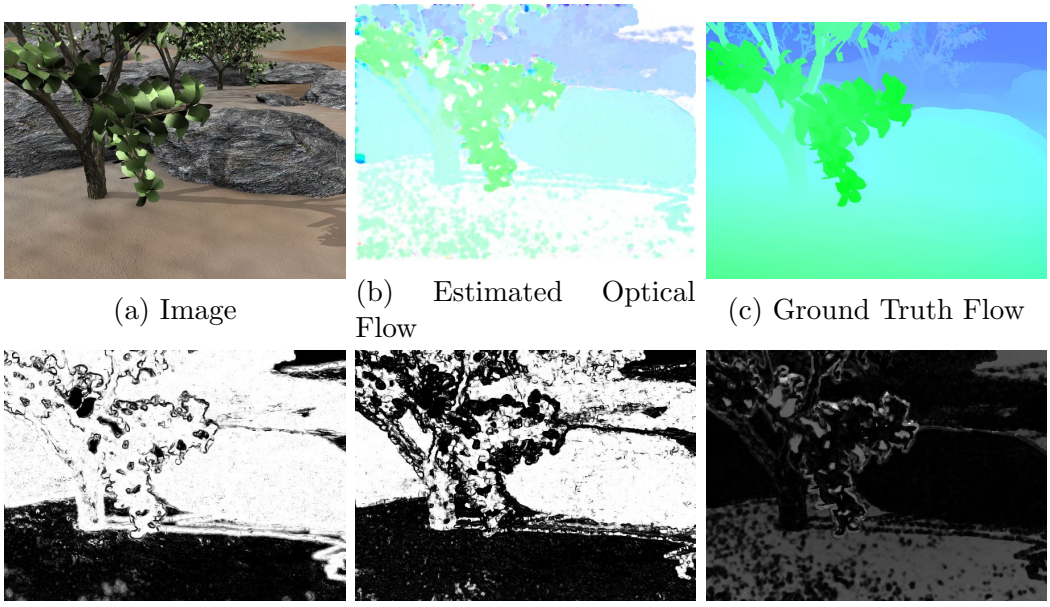
Training Criteria We evaluated confidence measures trained with EPE and MC used as the criteria for determining if a flow result is reliable when compared to the inertial data. We found the performance of EPE with a low threshold ('Inertial EPE 0.25') to be poor due to the errors in inertial motion estimation which frequently exceeded 0.25 pixels.

The tolerance of the training criteria to errors can be increased to allow improved average accuracy, as seen in figure 3-7. In this example, confidence measures trained using EPE with a larger threshold ('Inertial EPE 2.0'), or MC based measures, perform far better than 'Inertial EPE 0.25'. The consequence of increasing the leniency of the training criteria is an inability to detect small errors - so it becomes a matter of compromise depending on the required accuracy and the noise in the inertial estimates. We found the difference between using an EPE with a large error threshold and MC to be minimal, with both offering adequate results.

Scene Motion The 'Inertial Moving MC' confidence measure, which was trained from scenes containing motion, has a very similar performance to 'Inertial Static MC', which was trained from static scenes. This can be witnessed in figures 3-7g and 3-8g, where the two series follow each other closely.

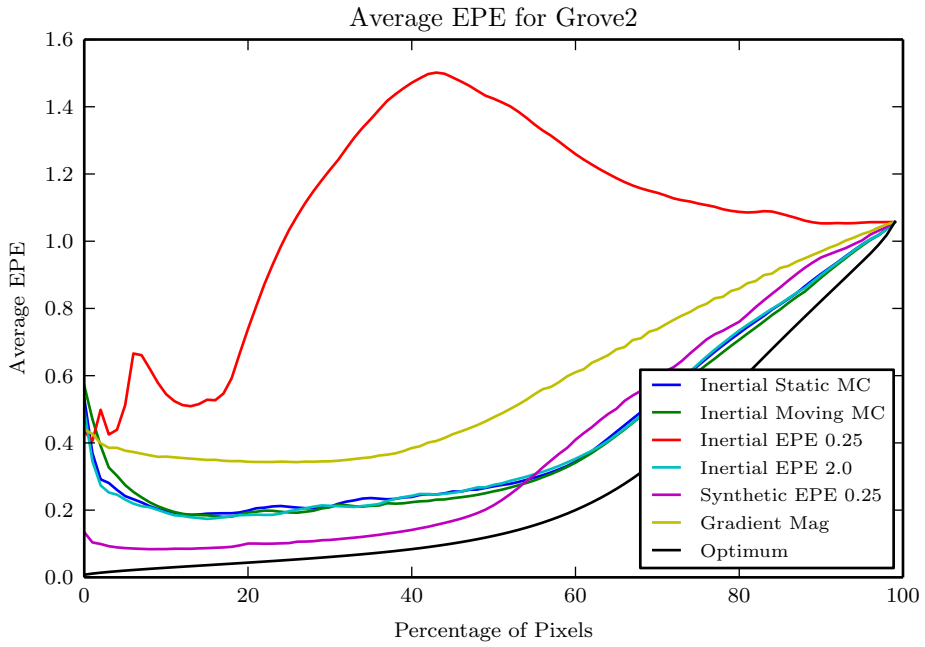
3.5.3 Evaluation on real data

The principle benefit of using inertial sensor data to train an optical flow confidence measure is the ability to easily obtain training data. The purpose of this experiment



(a) Image (b) Estimated Optical Flow (c) Ground Truth Flow

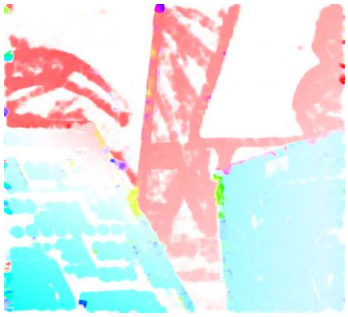
(d) ϕ - Inertial Static MC (e) ϕ - Synthetic EPE 0.25 (f) EPE



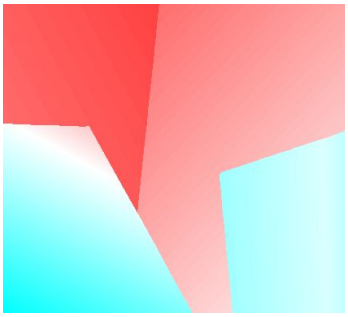
(g) Performance

Figure 3-7: 'Grove2' from [16]. This image contains detailed small-scale texture that can disappear at smaller scales. There are some errors in the inertial confidence measure where it assigns a high confidence around the edges of the branches which have high EPE. '*Inertial EPE 0.25*' performs poorly on this image. Confidence trained on our dataset with inertial data has slightly lower accuracy at detecting the most reliable regions, but performs better at detecting errors when compared to the synthetic source

(a) Image



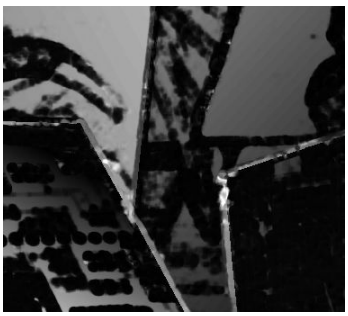
(b) Estimated Optical Flow



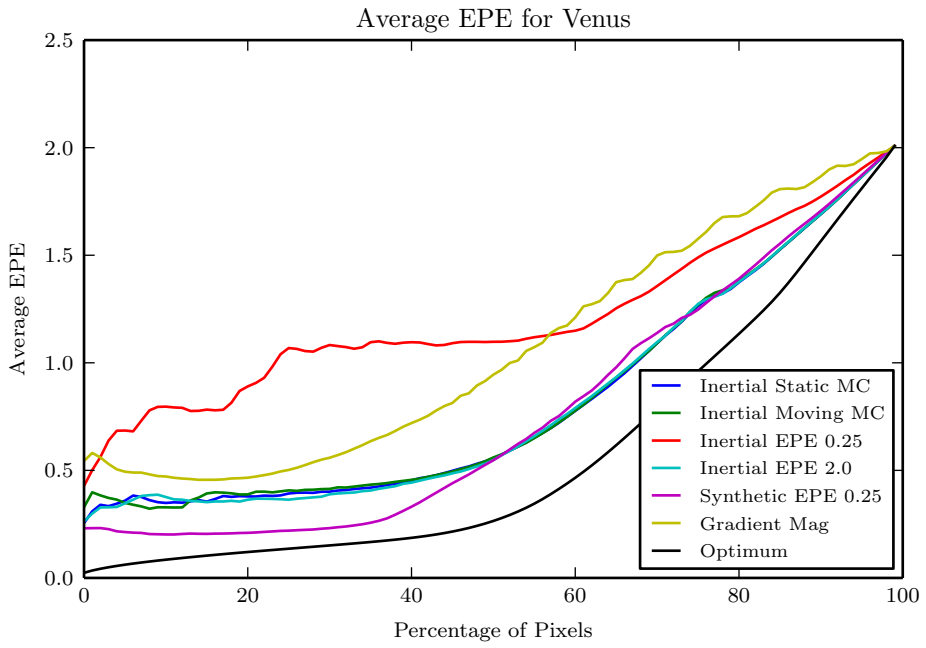
(c) Ground Truth Flow

(d) ϕ - Inertial Static MC

(e) ϕ - Synthetic EPE 0.25



(f) EPE



(g) Performance

Figure 3-8: 'Venus' from [16]. This sample contains planar motion with occluding regions around the objects. The homogeneous areas are correctly handled. The confidence measure trained on our inertial data has similar performance to the synthetic dataset for the low confidence regions

is to demonstrate the impact of training data selection on the performance of an optical flow confidence measure.

An evaluation dataset was built by randomly selecting 10 frames from each of 7 static video sequences, excluding any where the average motion magnitude was low. A sample of frames from each of these sequences is shown in figure 3-6. Given that the evaluation set was comprised of real images for which we had no inherent ground-truth data, we had to find an alternative. The inertial sensor data was not accurate enough to use as a basis for evaluation, so we instead found a reference motion field by fitting an homography transform to sparse feature correspondences, with a manual inspection step to check for accuracy of fit.

We used this dataset to evaluate two types of confidence measures. The first, 'Synthetic', was trained using Aodha's [26] synthetic images and corresponding flow fields resized and scaled to match the dimensions of the evaluation dataset. The second, 'Real', was trained using using inertial sensor data and real images from the 7 evaluation sequences, with frames used in the evaluation set excluded. In each case the feature set was as described in Section 3.3.2.

Figure 3-9 shows the result of this experiment. The plot shows a uniformly higher average EPE for the confidence measure trained using a synthetic dataset when compared to one trained using real images. This highlights the value of inertial sensors, as it has allowed us to produce a more accurate optical flow confidence measure as a result of being able to tailor the training data.

Sequence Specific

Here we show the results of each confidence measure on a single sequences. As before, we trained a confidence measure (labelled 'Sequence') using frames from the evaluation sequence, excluding any that were used in the evaluation set. The sequences we show are ones that differ most significantly from the others in the evaluation set, and therefore have the most to gain from a tailored confidence measure.

Figures 3-10 and 3-11 show the performance of various confidence measures on the sequences 'bushes' and 'shelves' respectively.

In the 'bushes' example, the optical flow result is very accurate as there is distinct detail throughout the image. A confidence measure trained using data only from this sequence shows a small improvement when compared to the one trained using only synthetic data. This example shows that inertial sensor data can be used to

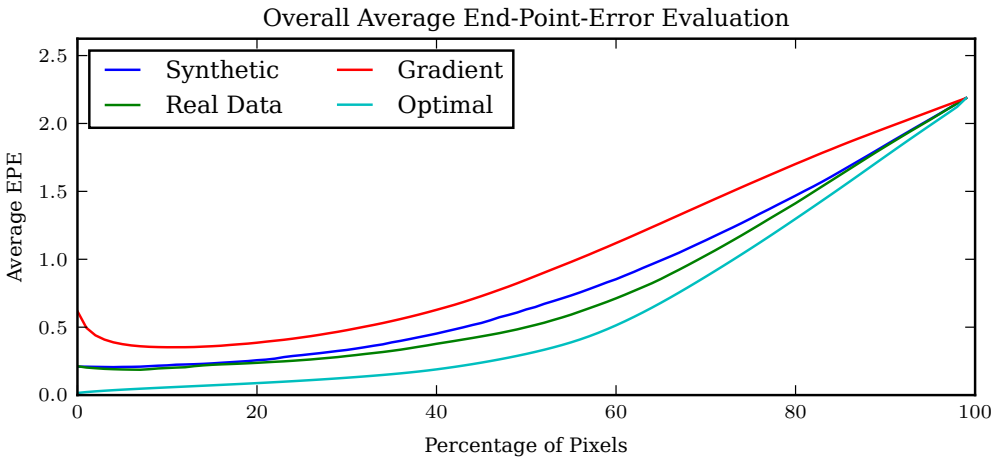


Figure 3-9: Evaluation of confidence measures trained using synthetic and real images. The images used for both training 'Real Data' and evaluation were taken from the same set of sequences, but no frames were used in both. The graph shows an average EPE lower for the confidence measure trained using real images.

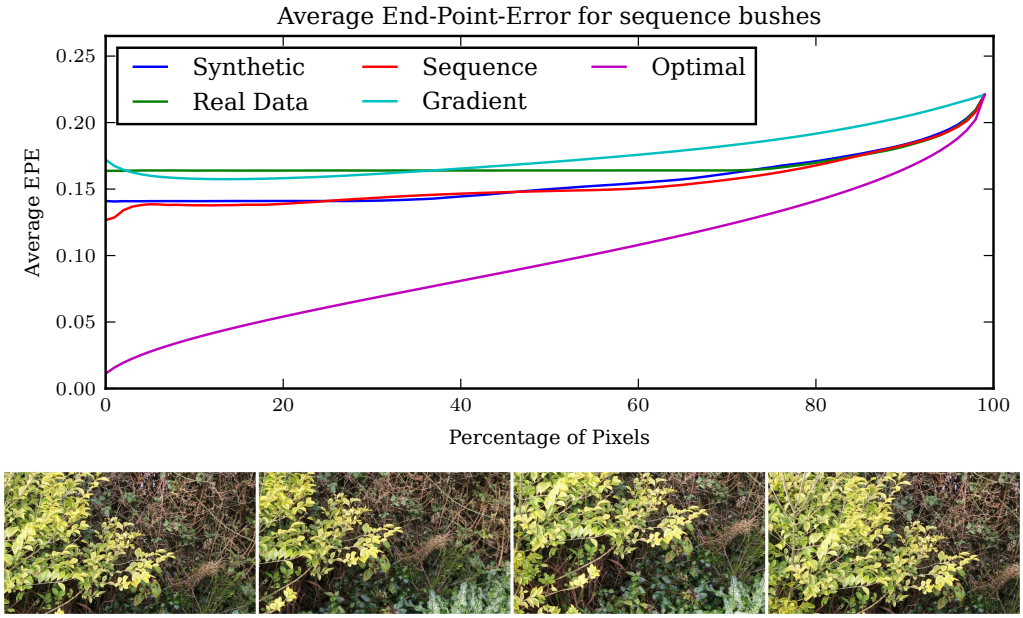


Figure 3-10: Confidence measure performance and sample frames for the 'bushes' sequence.

train an accurate optical flow confidence measure, even when the error margins are of subpixel accuracy. 'Real Data', which was trained using the larger set of real images, performs slightly worse than the synthetic dataset in this instance, but given the very small error margins, the difference is negligible. This reduction in performance is the result of a large proportion of the image being assigned the same high confidence value.

In the 'shelves' example we see a significant performance improvement when using training data tailored to the target scene. This is a result of lots of straight edges which make matching from one frame to the next difficult, leading to errors in the optical flow. The confidence measure trained just on this sort of scene is able to adjust the confidence for the edge areas.

This is where the ability to readily acquire training data tailored to the desired environment is of benefit. One option for improving the performance in such situations would be to use a more sophisticated feature set that takes into account both edge direction and flow direction. The downside of this approach is an increase in time taken producing a confidence image, both in terms of having to compute additional features and also in prediction. By training the confidence measure using tailored data we are able to avoid this and use a simple feature set that is applicable to real-time confidence estimation. The feature set we've used here takes an average of $11ms$ to compute the confidence for a 640x360 image.

3.6 Conclusion

We've presented a method for training an optical flow confidence measure using a combination of real images and inertial sensor data. We've evaluated it using both synthetic and real imagery and shown an improvement in performance in the latter case when using it in the target environment. This demonstrates the value of tailoring the training data, and therefore the value of using inertial sensors for this task. The ability to use inertial data as a basis for training optical flow confidence is important as it paves the way for future applications such as an online optical flow confidence measure that produces confidence estimates whilst simultaneously learning and adapting to the environment.

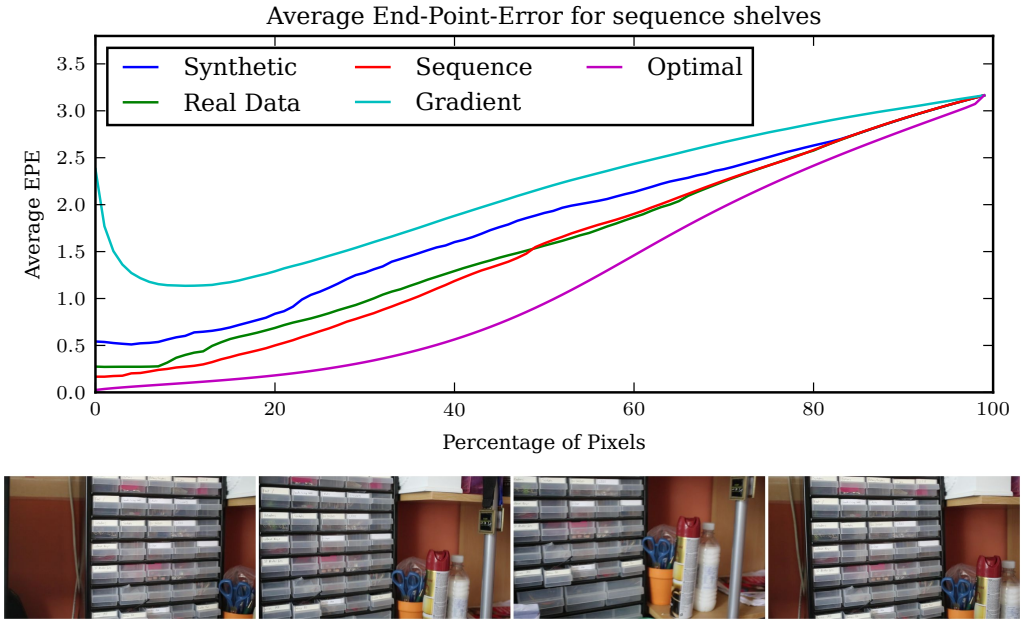


Figure 3-11: Confidence measure performance and sample frames for the 'shelves' sequence.

Chapter 4

Real-Time Inertial Motion Segmentation

The previous chapters of this thesis have concerned the estimation of the external motion of a camera and its relation to the anticipated apparent motion on the image sensor. We've discussed estimating image motion using optical flow, and noted its shortcomings. In response to these problems we developed a method of learning a confidence measure for optical flow from images and inertial data. In this chapter we apply these approaches to motion segmentation, with the motivation being to demonstrate how inertial sensors can simplify the problem of identifying and grouping moving regions in an image.

We conclude by presenting quantitative results that show that our simpler and more consistent method combining inertial sensor data with optical flow confidence outperforms the current state of the art in fast video segmentation by a factor of nearly 40 whilst achieving equivalent segmentation accuracy.

4.1 Introduction

Identifying scene motion from an image pair is fairly straightforward with a static camera; any significant movement in the image can be attributed to movement in the scene. If we consider the case of a moving camera capturing images of a moving scene, the problem becomes much more difficult - how do we differentiate between image motion caused by camera motion and image motion caused by motion in the scene? We use the inertial sensor data as prior on camera motion and compare the

predicted image motion with the observed image motion found using optical flow. Large differences between these quantities are likely to be the result of scene motion.

Our goal was to develop a motion segmentation scheme that, by taking advantage of inertial estimates of camera motion, enables detection of motion in images entirely within 2D image space and does not rely on building up a history of motion over several frames. Our motivation was to develop methods that are applicable to situations in which complex analysis is not feasible. We assume the camera motion to be restricted to the motions handled by our camera model (Chapter 2).

The goal of image segmentation is to divide an image into salient groups of neighbouring pixels, hopefully corresponding to some high-level feature. With just a single image, we are limited to making this decision based on prior knowledge of the scene, or by detecting boundaries between objects identified by variation in colour or illumination or other similar heuristics. Motion is a strong cue for identifying prominent objects; if a region undergoes coherent motion it is likely to correspond to some high-level feature, and so it stands to reason that we can improve our segmentation result by taking into account motion information.

In our case we are focused on binary segmentation, in which an image is divided into two disjoint regions, representing *static* and *moving* areas. The term *disjoint* is important, as we're not making any assumptions about the number of moving elements, and if there are several, it is likely their definitions will be disjoint. Defining a moving region is not always black and white; imagine a person stood still with their hand waving - should the whole body be labelled moving, as it belongs to an object exhibiting motion, or should the label be restricted to just the portions that are actually moving?

4.1.1 Problem Definition

A moving camera captures two images ($\mathbf{I}_0$ and $\mathbf{I}_1$) in quick succession. Inertial sensors mounted on the camera capture motion data which is used to produce an estimated *sensor flow field* $\mathbf{s}$ (see Chapter 2). The segmentation is performed on one of the images, $\mathbf{I}_0$, which is a function $\mathbf{I}_0 : \boldsymbol{\Omega} \rightarrow \mathbb{R}^c$ over the 2D domain $\boldsymbol{\Omega}$. The value of c depends on the number of channels in the image, with $c = 3$ for a colour image and $c = 1$ for grey-scale.

We define the motion segmentation problem as being the partitioning of $\boldsymbol{\Omega}$ into

2 disjoint areas:

$$\mathbf{\Omega} = \mathbf{\Omega}_S \cup \mathbf{\Omega}_M \tag{4.1}$$

with $\mathbf{\Omega}_S$ corresponding to the background or static regions and $\mathbf{\Omega}_M$ representing the moving bodies.

4.2 Background

Image segmentation on its own is a fairly ambiguous problem; for example, consider an image of a car. A valid segmentation could be the entire car, or it could be on a smaller scale and individually segment the wheels and windows, all of which are components of the car. In hierarchical image segmentation the wheels and window would be labelled as being part of the larger object. In our work we make no attempt at deducing a segmentation hierarchy.

Whilst we are concerned primarily with motion segmentation, which is a field of active research in its own right, it is still important to consider the key background theory of image segmentation, of which motion segmentation is a subset.

4.2.1 Non-Spatial Thresholding and Clustering

Thresholding can be used to form the most simple type of image segmentation. The segmentation is performed by labelling pixels based on their intensities or values after applying some other function to the image. In either case, thresholded segmentation schemes are very rudimentary and generally don't take spatial information into account.

A popular clustering technique is mean shift segmentation [31]. Each pixel is replaced with the mean of pixels in a window around it whose values are similar, typically determined as pixels with a Euclidian distance below some threshold.

K-Means is a clustering algorithm that is commonly applied to image segmentation. Its general principle is to define clusters in some multidimensional space, each with a *centroid* representing its position. Pixels are assigned to their nearest cluster centroid and the cluster centroids are then updated, repeating until convergence. The quality of a clustered segmentation result is very sensitive to the number of clusters.

4.2.2 Energy Minimisation Techniques

Energy minimisation techniques pose the segmentation problem as one of minimising an objective function. Energy minimisation allows for control over various aspects of the segmentation process, such as both the local properties of the segmentation and its global shape, perhaps with a preference towards a smooth segmentation boundary.

4.2.3 Contour-based Segmentation

Active contours or *snakes*[32] are contours that define segments in an image by their boundaries. In contrast to thresholding and clustering algorithms, active contours operate primarily in the spatial domain. Each 'snake' is a spline whose location is iteratively pulled towards features such as edges, and whose objective is to minimise the energy of several functions. Typically the energy functions encourage smoothness of the contour and draw the contour towards the desired image features.

The objective function of a snake is non-convex, which makes it very difficult to find a global optimum. Kass *et al.* [32] perform the optimisation using a finite differences approximation of the derivatives. In their method, the solution is highly dependent on the initialisation, requiring an initial boundary to be specified close to the desired segmentation.

Segmentation methods have been developed that define the partitioning as the level set of some high-dimensional function. The contour is characterised by the zero crossing of a surface, and is updated by varying the higher-dimensional function, allowing for changes in topology.

Global Region-based Segmentation

A region based method that searches for a globally optimal segmentation is desirable as it can take into account both the local features of an image as well as the overall structure of the segments.

Graph cuts have recently become a popular technique in computer vision. In this, an image is represented using a graph, which is an abstract representation of a set of interconnected nodes. A graph is defined as $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}})$, where $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{\mathcal{E}}$ denote the set of vertices and edges of $\boldsymbol{\mathcal{G}}$ respectively. In a *weighted* graph, each edge has a weight. An *s-t* graph is a weighted graph with two additional nodes, the

source s and the *sink t*. A *cut* refers to a partitioning of the graph, and in the case of an s-t graph, a cut partitions the graph such that there is no flow from the source to the sink. Each cut has an associated cost that is the sum of the weights of the edges removed. A *minimum cut* of an s-t graph is the cut that separates the source from sink with the minimum sum of weights of the removed edges.

Segmentation using graph-cuts can be performed at either the pixel level, or by first grouping an image into salient blocks or 'superpixels'. In each case, a graph is constructed by connecting each of these low-level structures to its neighbours as well as to both the source and sink. The connections to the source and sink are the focus of the segmentation, with each representing either moving or static regions. After a cut of the graph has been made, each node will be connected to either the source or the sink but not both.

The weights between neighbouring nodes are chosen such that the cost for making a cut is lowest at likely boundaries between regions so that a segmentation will prefer to follow edges rather than cut across the centre of a region. This is shown in Figure 4-1.

The weights between each node and the source and sink node are the key factor that dictates the final labelling. These costs, often referred to as the unary costs, set the affinity of each pixel or superpixel to being either moving or static. The source and sink weights could come from user input in the form of a highlighted area specifying the object to be segmented as in [33], where the target image is annotated with scribbles to indicate foreground and background regions, or they could, as in the case of our work, be set automatically according to some prior knowledge such as motion. Extensions for multi-label segmentation using graph cuts have been investigated but are beyond the scope of our work so are not covered in this thesis.

4.2.4 Motion Segmentation

There are several types of image motion that have been considered as part of motion segmentation. A common focus is the motion caused by parallax as a camera translates between capturing images of a scene with objects at different depths, causing each object to have a velocity that is scaled according to the depth. Another case is when the camera remains static whilst some element of the scene moves. Finally, there is the case that we target, where a moving camera captures a moving scene.

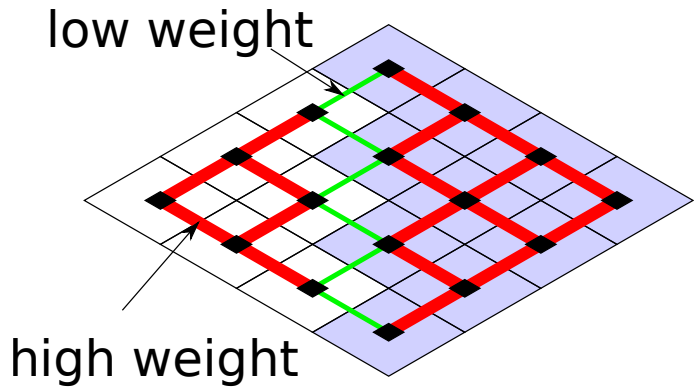


Figure 4-1: Example graph from an image. The grid represents a 4x4 block of pixels, connected in a 4 neighbourhood. Each pixel is additionally connected to the source and sink nodes which are not shown here. The red lines have high edge costs as the pixels they connect are the same colour. The edges that span the border between the white and the blue regions have a low edge cost due to the gradient. The minimum cut of this graph would remove the green edges. After the cut has been made one set of pixels will be connected to just the source node, and the other set to just the sink node.

Our work is an unsupervised method, so we do not cover methods such as [34] that rely on human interaction to perform motion segmentation.

The benefit of having several frames of images has been evident in many past works. Zhou *et al.* [35] look at the motion history of SIFT[13] features tracked over multiple frames and use RANSAC to identify points with anomalous motion, then perform a crude segmentation via thresholding to find the moving regions, without taking into account homogeneous regions that are unlikely to have SIFT features. Similarly, Brox *et al.* [36] analyse the long term trajectories of image features to group points by their motion, but fail to group the centre of a solid colour to the motion at its boundaries. Ochs and Brox [37] extend the approach to handle non-translational motion such as rotation or scaling. Over a large number of frames (in one example they state 800) they are able to reliably group sparse points according to their motion. One of the goals of our work is to use inertial sensor data to relax requirements such as the need for a long history of images to deduce motion, and so we focus on segmentation from a single pair of images.

Cremers and Soatto [38] describe motion segmentation of a moving scene captured using a moving camera as a 'chicken and egg' problem. They use two consecutive frames to segment the image plane into two regions with the same parametric

motion, with the assumption of small magnitude motion and constant brightness. The segmentation is performed using a level set method. According to the authors their method does not scale well to multiple moving regions. Dinh and Medioni [39] use tensor voting and graph-cuts to simultaneously estimate the motion and segmentation from a pair of frames. Their motion estimation method is very slow, taking several minutes. Their graph-cut segmentation starts by labelling each pixel as either foreground, background or undefined, and sets edge costs based on the assumption that pixels of similar colour are likely to have similar motions, and that the moving region is much smaller than the static one.

Xu *et al.* [40] learn the relationship between the motor signals driving a robot and the motion captured by a camera. They then track sparse features and identify moving parts as those which differ from the predicted motion according to the motor signals. They experiment with both an active contour and graph-based method for segmentation.

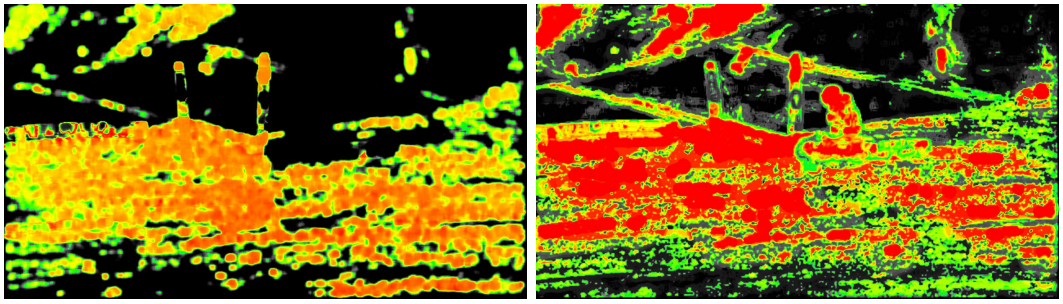
Unger *et al.* [41] perform a joint motion estimation and segmentation of an image pair. They search for regions with motion that can be modelled by the same parametric transformation, and alternate between estimating the motion and updating the segmentation for all regions. For scenes with non-rigid motion, they assign a pre-calculated optical flow field as the motion for one label, which causes portions of the image with continuous flow to be assigned to that label if no parametric motion can describe it.

The work of Papazoglou *et al.* [42] can be considered the state of the art of motion segmentation where run-time performance is an important consideration. They achieve this result by looking for discontinuities in the optical flow field to produce a rough segmentation. A superpixel-based graph is then cut over the entire sequence to get the result. Their 'fast' method cites results of 0.5 seconds per frame (at 400x225 resolution), but this is only after excluding the time taken to compute optical flow.

The method we've developed is not unique in utilising inertial sensors to detect motion. Lobo *et al.* [43] combine inertial sensors (accelerometer, gyroscope and magnetometer) with image and depth data from a Kinect camera. They produce a 3D voxel grid of the scene over many frames, and consider 3D positions with consistent readings as stationary. Aside from their requirement of a specialised depth camera, their analysis requires many frames.

(a) Image **I** (b) Optical Flow **u** (c) Sensor Flow **s**

Figure 4-2: Input to motion segmentation from sequence 'snowboarder'. In this example the camera is panning whilst the snowboarder moves. The flow in (b) and (c) is visualised using the colour coding described in Chapter 3.



(a) Correlation (b) Confidence

Figure 4-3: Motion Analysis. (a) shows the correlation between optical flow and motion estimate from inertial sensors. (b) is the optical flow confidence produced by our inertial dataset. The colourmap used is described in Figure 2-9.

4.3 Motion Segmentation using Inertial Sensors

In this section we describe our method for partitioning an image into static and moving regions, whilst referring to example images.

The input to our algorithm is an optical flow field **u**, an estimate of image motion **s** computed using inertial sensor data as described in Chapter 2, and the first frame of the image pair, **I**. These are shown in Figure 4-2.

We estimate the optical flow confidence $\boldsymbol{\phi}$ using the method described in Chapter 3. The confidence measure is trained using inertial data for the algorithm used to produce the flow field **u**. The confidence $\boldsymbol{\phi}$ tells us where we can expect **u** to be reliable, which is essential for us to be able to determine whether differences in **u** and **s** are due to scene motion or errors in the optical flow. The utility of this measure is apparent in Figure 4-2, where there are several large featureless regions for which no flow is detected. This effect propagates into the motion correlation in Figure 4-3a,

where we see low correlation for many regions. The confidence in Figure 4-3b tells us that the optical flow computed on the snowboarder should have been reliable, so the fact that the correlation is low in the region suggests it represents scene motion.

4.3.1 Graph Weights

We use graph-cut as the basis for our segmentation method. We experimented with both dense pixel-wise segmentation and a sparser superpixel segmentation and found the latter to be better suited to our application as it was much faster to cut the graph. The superpixels are produced using the fast GPU method of [44]. Full details of the performance of this step can be found in Section 4.4.2. In the following sections we describe how we build up the graph and the weights associated with each edge. We calculate the minimum cut of the graph using the method of Delong *et al.* [45], which incorporates label costs.

The goal of the segmentation is to assign to each superpixel $p \in \boldsymbol{\mathcal{P}}$ a label $f_p \in \{0, 1\}$. The segmentation is essentially an energy minimisation of the following objective function:

$$E(f) = \underbrace{\sum_{p \in \boldsymbol{\mathcal{P}}} D_p(f_p)}_{\text{data cost}} + \underbrace{\sum_{pq \in \boldsymbol{\mathcal{N}}} w_{pq}(f_p, f_q)}_{\text{smooth cost}} + \underbrace{\sum_{L \subseteq \boldsymbol{\mathcal{L}}} h_l \cdot \delta_L(f)}_{\text{label cost}} \tag{4.2}$$

where

$$\delta_L(f) = \begin{cases} 1 & \exists p : f_p \in \mathbf{L} \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

and $\mathbf{L}$ is the label subset.

Data Costs

The data or unary costs are the weights between each pixel and the sink (static) and source (moving) nodes. The basis of our method is to assign weights to the sink and source only for those regions where we predict the optical flow to be reliable. The remaining areas, which can often constitute a large proportion of the image, are left as undefined, with no affinity towards either moving or static. The end result is that undefined regions are likely to be combined with neighbouring areas which do have an inclination towards a particular label. For pixels with high confidence,

edge	weight	where
$\{p, q\}$	$w_{p,q}$	$\{p, q\} \in \boldsymbol{\mathcal{N}}$
$\{p, S\}$	$\phi_p C_p$	$p \in \boldsymbol{\mathcal{B}}$
	0	$p \in \boldsymbol{\mathcal{M}}$
$\{p, T\}$	0	$p \notin \boldsymbol{\mathcal{B}}$
	$\phi_p(1 - C_p)$	$p \in \boldsymbol{\mathcal{B}}$

Table 4.1: Edge weights (costs) for nodes representing pixels (p, q), source S (moving) and sink T (static).

a low weight to the source node is assigned if the pixel has a high error, and a high weight if the pixel has low error. The weights to the sink node are the opposite.

We define $\boldsymbol{\mathcal{B}}$ as the set of pixels that are very likely to correspond to static world elements. We find $\boldsymbol{\mathcal{B}}$ by thresholding:

$$\mathcal{B}_i = (\phi_i C_i) > \mathcal{B}_{\text{th}} \tag{4.4}$$

Similarly, we define $\boldsymbol{\mathcal{M}}$ as the set of pixels that have a high chance of corresponding to moving objects:

$$\mathcal{M}_i = (\phi_i(1 - C_i)) > \mathcal{M}_{\text{th}} \tag{4.5}$$

In our experiments we set $\mathcal{B}_{\text{th}} \geq \mathcal{M}_{\text{th}}$. The weights used for the edges of the graph are shown in Table 4.1.

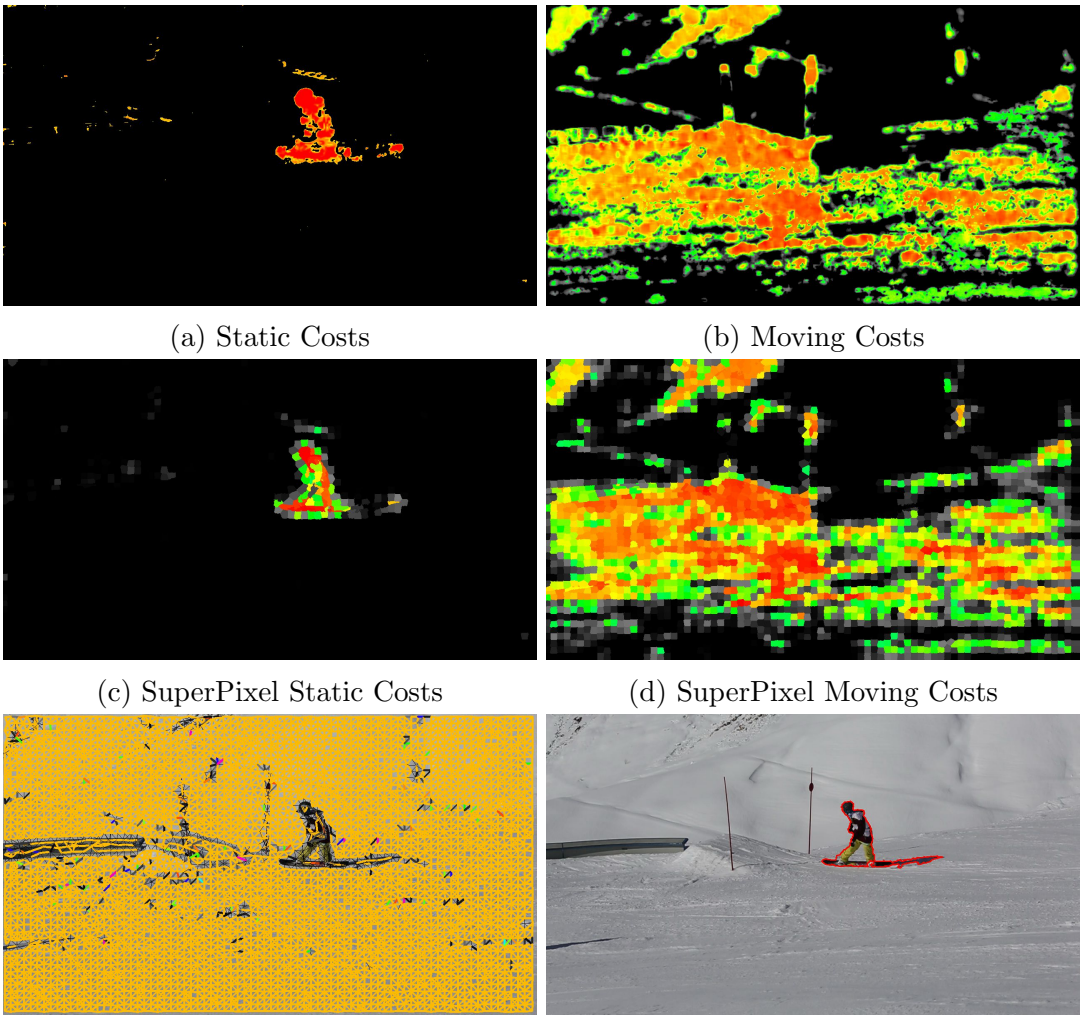
The data costs are calculated for individual pixels, and then averaged within each superpixel. This step is shown in Figures 4-4a and 4-4c.

Edge Costs

We set the edge cost between two adjacent superpixels as a function of the difference between the average RGB colour and the Euclidean distance. As in the works of *e.g.* [42, 46], these are constrast-modulated Potts potentials:

$$w_{p,q} = \frac{\exp(-\beta col(p, q)^2)}{dis(p, q)} \tag{4.6}$$

where $col(p, q)$ and $dis(p, q)$ is difference between the average RGB intensity and Euclidean distance between centres of superpixels p and q respectively. β is a pa-



(a) Static Costs

(b) Moving Costs

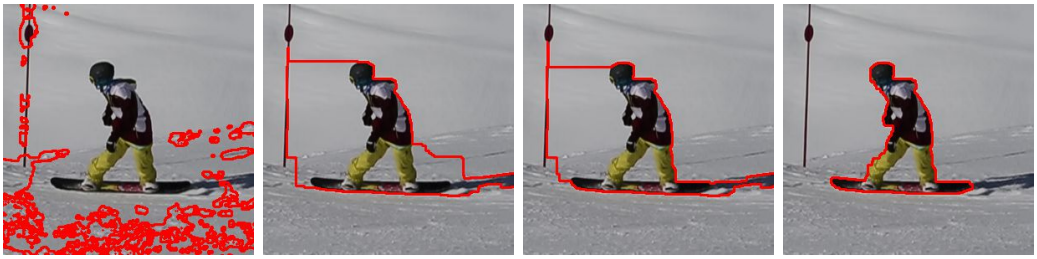
(c) SuperPixel Static Costs

(d) SuperPixel Moving Costs

(e) Edge Costs

(f) Segmented Image

Figure 4-4: Graph weights for a sample image. (a) and (c) show the costs for labelling each pixel and superpixel respectively as static, whilst (b) and (d) show the same for moving. (e) is the edge weights between neighbouring superpixels, with a thick yellow line representing high cost and a thin black line low cost. (f) is the labelled segmentation result.



(a) $h_L = 0$ (b) $h_L = 5$ (c) $h_L = 10$ (d) $h_L = 15$

Figure 4-5: Effect of label costs. As h_L increases, the cost of having a long border increases, so the segmentation is drawn towards shorter solutions.

rameter that can be set adaptively according to the average ratio of the Euclidean norm of colour distance and centre distance.

Figure 4-4e shows an example of these edge costs. The connections with low weights are indicated by thin black lines (in contrast to the thicker yellow lines), and these edges cross strong edges in the images.

Label Costs

A label cost is the cost of two adjacent nodes having different labels. Increasing the label cost encourages a smooth segmentation result as the graph is cut in such a way to minimise the number of instances where two neighbouring pixels have different labels. Figure 4-5 demonstrates the effect of the label costs, where it can be seen that as the label cost increases, the segmentation boundary length becomes shorter.

4.3.2 Process Overview

Algorithm 2 outlines our motion segmentation method. It shows how we combine the techniques developed in Chapters 2 and 3 to build our motion segmentation technique.

4.4 Evaluation

The direct evaluation of our method is difficult due to the requirement of inertial sensor data. This means that we are not able to simply run our method against the data used by previous publications and make a direction comparison with their published results. Instead, we must run the methods on our data and compute new

Input: Set of images from video sequence - $\boldsymbol{\mathcal{F}}$, inertial sensor data
Output: Motion Segmented Images
Initialisation:
Align sensor data with images using the method described in Chapter 2.
Compute $\boldsymbol{\mathcal{H}}$, which is a set of homography matrices that describe the relative motion from each frame to the next.
for $\mathbf{F}_i \in \boldsymbol{\mathcal{F}}$, $\mathbf{H}_i \in \boldsymbol{\mathcal{H}}$ **do**
  $\mathbf{u}$ = OpticalFlow($\mathbf{F}_i$, $\mathbf{F}_{i+1}$)
  $\mathbf{s}$ = SensorFlow($\mathbf{H}_i$)
  $\mathbf{C}$ = MotionCorrelation($\mathbf{u}$, $\mathbf{s}$)
  $\boldsymbol{\phi}$ = OpticalFlowConfidence($\mathbf{F}_i$)
  $\boldsymbol{\Omega}_M$ = MotionSegment($\mathbf{C}$, $\boldsymbol{\phi}$)
end

Algorithm 2: Motion Segmentation process overview

results, a step that is far from trivial due to the complexity involved in implementing these works and the frequent unavailability of source code.

In this section we evaluate our method against the work of Papazoglou *et al.* [42], both in terms of the segmentation accuracy and also the time taken to achieve the result. We chose to evaluate against this work for several reasons. Firstly, the authors have published their source code, satisfying the above mentioned difficulty. Secondly, their work, which represents the state-of-the-art in motion segmentation, includes comparisons of their method against others such as [36], and have shown a comparable segmentation accuracy at greatly reduced run-times. In this section we show that our method is faster than Papazoglou's, and therefore the body of literature, whilst maintaining parity in terms of segmentation accuracy. We did test our sequences with the implementation of Brox *et al.* [36] but found it to have a run time in the order of 30 minutes per frame, making it not at all suitable for real-time analysis.

We performed our evaluation on five sequences: 'snowboarder', 'runner', 'balls', 'car' and 'horses'. These sequences are captured by a rotating camera and feature both rigid and non-rigid scene motion, as well as multiple moving objects.

The results presented include different configurations of each method. The details of these configurations are as follows:

FVS Brox Flow is the work of [42] using the costly optical flow method of [47], and runs at a resolution of 400x225.

FVS basic flow is as 'FVS Brox Flow' except it uses the same optical flow process and parameters as our method, which is much quicker to produce but less accurate.

Ours 1280 1/2 motion is our method with 1280x720 source images and motion analysis (optical flow, optical flow confidence, motion correlation) performed at half scale (640x360).

Ours 1280 1/4 motion is as 'Ours 1280 1/2 motion' except motion analysis is performed on 320x180 images.

Ours 640 1/2 motion is our method with 640x320 images and 320x180 motion analysis.

Ours SIFT features is our method with camera motion found by fitting an homography to RANSAC filtered SIFT feature correspondences. The source images are 1280x720 and motion analysis (optical flow, motion correlation, homography fitting) is performed on half scale (640x360) images. When benchmarking this method we make use of the fact that whilst two sets of features are required to compute the motion, the result of one of these can be cached for the subsequent frame, resulting in reduced computation time.

4.4.1 Segmentation Accuracy

We evaluated the accuracy of our method by comparing the estimated motion segmentation $\mathbf{\Omega}_M$ with hand labelled ground truth image $\mathbf{\Omega}_{\text{gt}}$. Not all frames of each sequence are labelled due to the high cost associated with manually tracing around the moving object. Instead we labelled an evenly distributed sample of frames, for example every 5th frame. The ground truth data is slightly problematic as a result of errors in human-input and the ambiguity of labelling motion. We only label the significant subject of interest, but there are often small motions in the scene that might be detected by our segmentation method, including shadows or reflections of our subject, and there is also the possibility that not all of the subject is moving.

Despite these shortcomings in the ground truth data we can still use it to assess how well a method performs at detecting moving objects.

Evaluation Criteria

We evaluate segmentation accuracy using two criteria; absolute accuracy of a segmentation result, as well as its ability to pick out the general location of a moving object. In the former, we compare the size of both the intersection and union of the segmentation result $\mathbf{\Omega}_M$ and ground truth labelling $\mathbf{\Omega}_{\text{gt}}$ to get the accuracy A:

$$A = \frac{\sum \mathbf{\Omega}_M \cap \mathbf{\Omega}_{\text{gt}}}{\sum \mathbf{\Omega}_M \cup \mathbf{\Omega}_{\text{gt}}} \tag{4.7}$$

The optimal result of $A = 1$ is found when $\mathbf{\Omega}_M = \mathbf{\Omega}_{\text{gt}}$. If the ground truth segmentation is contained entirely within $\mathbf{\Omega}_M$ but takes up half the area then the accuracy will be 0.5.

This leads onto the second criterion, in which we determine if the moving object(s) have been detected, even if their outline is not perfect. We do not want a segmentation that labels the entire image as moving to score as well as one that only labels the moving object. Instead, we allow for a small extension beyond the ground truth borders to score as well as the optimal result, with the amount of extension being relative to the size of the ground truth object. We define a second measure of performance, the 'reasonable coverage' accuracy measure:

$$C_\alpha = \frac{\sum \mathbf{\Omega}_M \cap \mathbf{\Omega}_{\text{gt}}}{\sum \mathbf{\Omega}_{\text{gt}} + max(\sum \mathbf{\Omega}_M \cap \overline{\mathbf{\Omega}_{\text{gt}}} - \alpha \sum \mathbf{\Omega}_{\text{gt}}, 0)} \tag{4.8}$$

where α is a parameter that allows the area of $\mathbf{\Omega}_M$ that is outside of $\mathbf{\Omega}_{\text{gt}}$ ($\mathbf{\Omega}_M \cap \overline{\mathbf{\Omega}_{\text{gt}}}$) to be a fraction α larger than the size of $\mathbf{\Omega}_{\text{gt}}$ before it has a negative impact on the accuracy. For example, when $\alpha = 0.2$, $C_{0.2} = 1$ is possible as long as $\mathbf{\Omega}_{\text{gt}}$ is fully contained within $\mathbf{\Omega}_M$, i.e $\mathbf{\Omega}_M \cup \mathbf{\Omega}_{\text{gt}} = \mathbf{\Omega}_M$, and the size of extra coverage, i.e $\mathbf{\Omega}_M \cap \overline{\mathbf{\Omega}_{\text{gt}}}$, is not more than 20% of the size of $\mathbf{\Omega}_{\text{gt}}$. C_∞ represents the proportion of $\mathbf{\Omega}_{\text{gt}}$ that is covered by $\mathbf{\Omega}_M$ irrespective of the inclusion of any static regions.

In addition to presenting the average segmentation accuracy for a sequence, we also find the proportion of frames with an accuracy exceeding a set of thresholds: 60% ($A \geq 0.6$), 70% ($A \geq 0.7$), 80% ($A \geq 0.8$) and 90% ($A \geq 0.9$). This gives an indication of the performance of the method for different error margins.

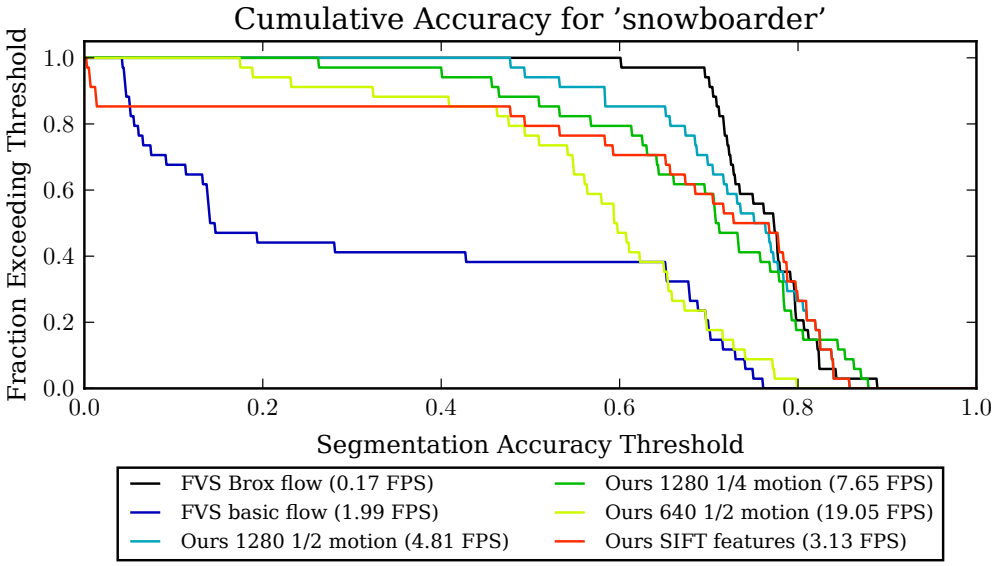


Figure 4-6: Cumulative threshold performance of each method in several configurations for the 'snowboarder'. The graph shows the proportion of frames with a segmentation accuracy A exceeding a given threshold

Results

A comparison of our method with and Papazoglou's in a variety of configurations is shown in figures 4-11, 4-12, 4-13, 4-14 and 4-15. These figures feature the cropped (according to ground truth mask) segmentation result of a subset of evenly distributed frames, and quantitative accuracy results for the entire sequence (where ground truth data is available). We now discuss each test sequence and the results.

Snowboarder A snowboarder rides across the snow and then slides along a box, as shown in Figure 4-11. This sequence has 164 frames of which 34 have manually-labelled ground truth. The cumulative performance thresholds are shown in Figure 4-6. This sequence performed particularly well due to a general lack of complicated background structure and a strong contrast between subject and background.

The accuracy scores do not necessarily reflect the functional performance on this sequence as the subjects shadow is frequently included in the moving label. In this case, $C_{0.2}$ gives a better indication of the accuracy of the match as it allows for some excess labelling.

Papazoglou's method with our basic flow performs particularly badly on this

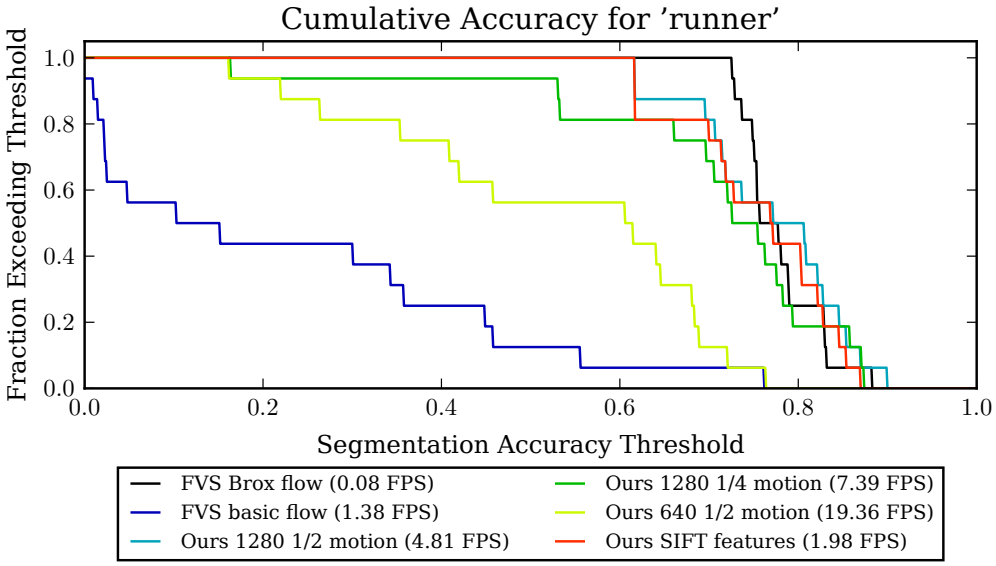


Figure 4-7: Cumulative threshold performance of each method in several configurations for the 'runner'. The graph shows the proportion of frames with a segmentation accuracy A exceeding a given threshold

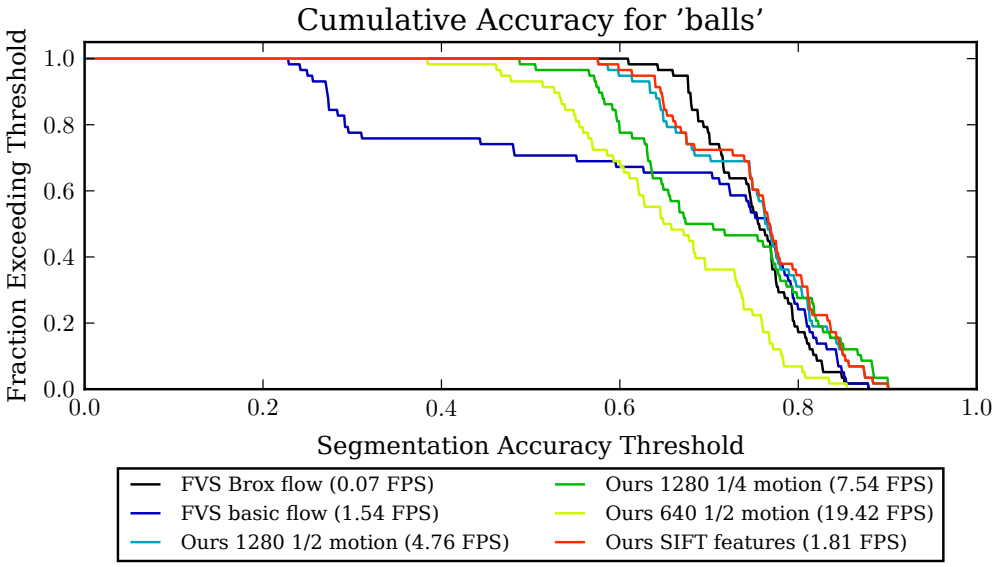


Figure 4-8: Cumulative threshold performance of each method in several configurations for the 'balls'. The graph shows the proportion of frames with a segmentation accuracy A exceeding a given threshold

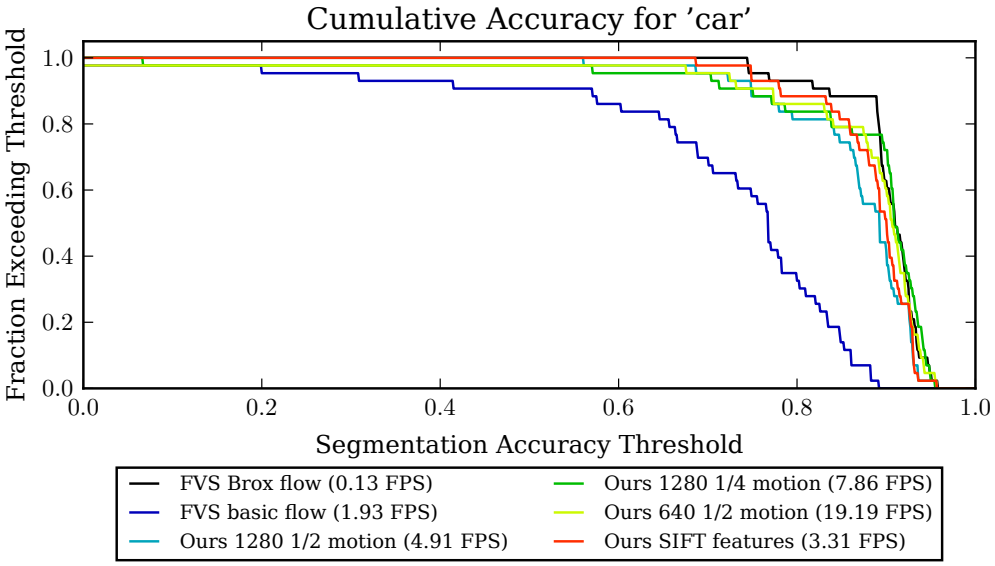


Figure 4-9: Cumulative threshold performance of each method in several configurations for the 'car'. The graph shows the proportion of frames with a segmentation accuracy A exceeding a given threshold

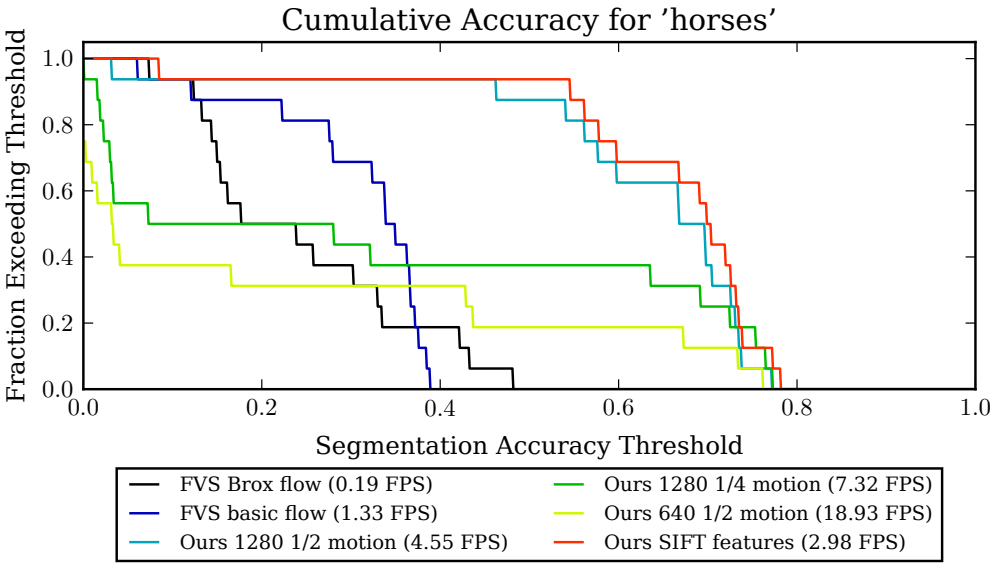


Figure 4-10: Cumulative threshold performance of each method in several configurations for the 'horses'. The graph shows the proportion of frames with a segmentation accuracy A exceeding a given threshold

sequence, with an average segmentation accuracy of less than 30%. This sequence exhibits some failure modes in motion fitting, causing several frames to completely fail for 'Ours SIFT features'. An analysis of these failures can be found in Section 4.4.3.

Runner

In this sequence the subject runs across the scene and jumps, as shown in Table 4-12. There are 110 frames, of which 16 have ground truth. The cumulative performance thresholds are shown in Figure 4-7. We filmed this sample using a 35mm lens. The scene was reasonably close to the camera, so we used less strict motion correlation parameters than with 'snowboarder', to allow for variations in depth.

This example is particularly challenging, with the motion being much less rigid than 'snowboarder'. Our ground truth includes the whole of the subject, even though some parts are mostly static, for example the foot whilst planted on the ground. The segmentation border extends beyond the edge of the runner in some cases due to the relatively poor contrast between the dark clothing and the background.

Papazoglou's method with our basic flow struggles with the complex motion, achieving an average segmentation accuracy of less than 25%. By contrast our fast configuration 'Ours 640 1/2 motion' achieves a respectable average accuracy of over 60% whilst being more than 200 times faster than 'FVS Brox flow'.

Balls

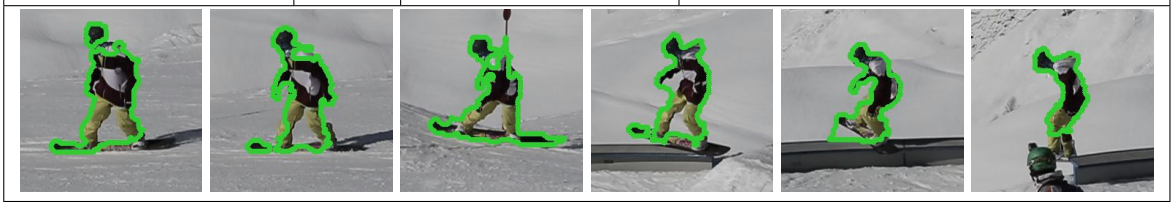
Two balls are thrown across the grass as the camera moves, as shown in figure 4-13. There are 66 frames, of which all have ground truth, as the balls are very quick to manually label. The cumulative performance thresholds are shown in Figure 4-8. The lens used had a focal length of 35mm and the scene had a wide range of depths, from the grass in the foreground to the bushes in the background. To cope with this we had to relax the motion correlation parameters such that a wider range of motion scales could be considered to be correlated with the camera motion.

On the whole the performance for this sequence was good. Papazoglou's method with our cheap optical flow ('FVS basic flow') performed well on this sequence, a result of rigid moving objects. Our fastest configuration, 'Ours 640 1/2 motion', scored less than the other methods as the ball's shadow was included in the segmentation result in many frames. Given the relative size of ball and shadow, this had

Hand-labelled Ground Truth Segmentation Mask

Cropped Segmentation Results

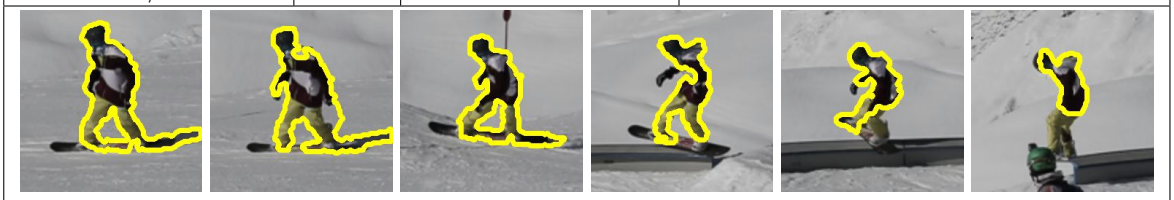
		Accuracy			Frames exceeding threshold			
Method	FPS	A	C_∞	$C_{0.2}$	0.6	0.7	0.8	0.9
FVS Brox flow	0.168	0.761	0.803	0.803	100.0%	94.1%	20.6%	0.0%



FVS basic flow	1.986	0.342	0.864	0.399	38.2%	17.6%	0.0%	0.0%

Ours 1280 1/2 motion	4.808	0.728	0.853	0.819	85.3%	67.6%	26.5%	0.0%

Ours 640 1/2 motion	19.052	0.572	0.623	0.622	47.1%	17.6%	0.0%	0.0%



Additional Results

Ours 1280 1/4 motion	7.651	0.687	0.748	0.737	79.4%	58.8%	17.6%	0.0%
Ours SIFT features	3.131	0.626	0.807	0.698	70.6%	58.8%	26.5%	0.0%

Figure 4-11: Evaluation of our method against Papazoglou's 'Fast Video Segmentation' (FVS) in a variety of configurations for the sequence 'snowboarder'. The thumbnails show cropped segmentation results for a subset of frames, with numerical results computed over all frames with ground truth data. The final two rows in the 'Additional Results' section of the table are not pictured. The accuracy criteria used is explained in Section 4.4.1.

Hand-labelled Ground Truth Segmentation Mask


Cropped Segmentation Results								
		Accuracy			Frames exceeding threshold			
Method	FPS	A	C_∞	$C_{0.2}$	0.6	0.7	0.8	0.9
FVS Brox flow	0.082	0.779	0.861	0.861	100.0%	100.0%	25.0%	0.0%
								
FVS basic flow	1.382	0.226	0.245	0.245	6.2%	6.2%	0.0%	0.0%
								
Ours 1280 1/2 motion	4.814	0.769	0.898	0.878	100.0%	81.2%	50.0%	0.0%
								
Ours 640 1/2 motion	19.362	0.520	0.606	0.591	56.2%	12.5%	0.0%	0.0%
								
Additional Results								
Ours 1280 1/4 motion	7.391	0.700	0.825	0.797	81.2%	68.8%	18.8%	0.0%
Ours SIFT features	1.981	0.755	0.882	0.866	100.0%	75.0%	43.8%	0.0%

Figure 4-12: Evaluation of our method against Papazoglou's 'Fast Video Segmentation' (FVS) in a variety of configurations for the sequence 'runner'. The thumbnails show cropped segmentation results for a subset of frames, with figures computed over the entire sequence. The final two rows in the 'Additional Results' section of the table are not pictured. The accuracy criteria used is explained in Section 4.4.1.

a significant impact on the accuracy result. The $C_{0.2}$ accuracy result better reflects the performance as this allows for the extra overlap. With this measure we achieve a higher average accuracy when compared to 'FVS basic flow'.

Car

A car drives along a road as the camera pans in the opposite direction, as shown in Figure 4-14. The sequence is 96 frames long and all have ground truth labelling. The cumulative performance thresholds are shown in Figure 4-9. This sequence gave particularly good results, owing to the car's rigid structure and large uniform shape. Even 'FVS basic flow' achieves reasonable results, a result of the simple to interpret motion.

Horses

This is the most challenging of our sequences. Three horses run along the grass as the camera pans, as shown in Figure 4-15. One of the horses is visible at the very start and is then out of scene until the final frames. The horses represent a very difficult case of non-rigid motion with complex trajectories particularly in the legs. This sequence is 160 frames long, of which 16 have been labelled. The cumulative performance thresholds are shown in Figure 4-10.

Papazoglou's method struggled with this sequence, including large sections of the background in their segmentation mask. Our lower resolution, faster methods also faired badly, as the low resolution optical flow field was not of sufficient detail. Our high resolution configuration, 'Ours 1280 1/2 motion', achieved an average accuracy of over 60%. This is a good result for this sequence, as the long legs of the horse make the graph-cut segmentation likely to take a 'short-cut' and include some of the background.

4.4.2 Segmentation Performance

In this section we assess its suitability for real-time applications by performing a series of benchmarks. The results quoted here were produced using modest hardware: an Intel(R) Core(TM) i5-2500K CPU @ 3.30GHz and an NVidia GTX 560. We implemented our method using Python with calls to compiled code (both our own and from libraries such as OpenCV) used where appropriate. This is comparable to

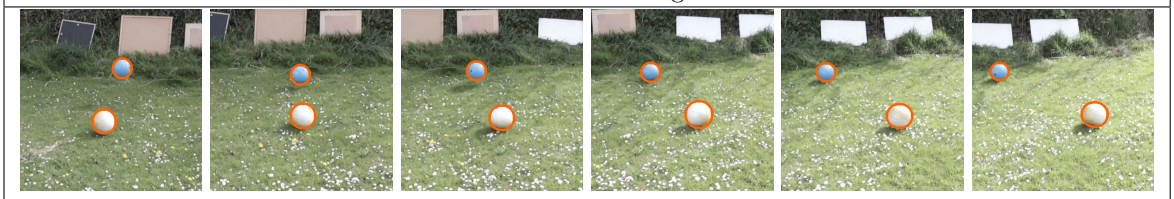
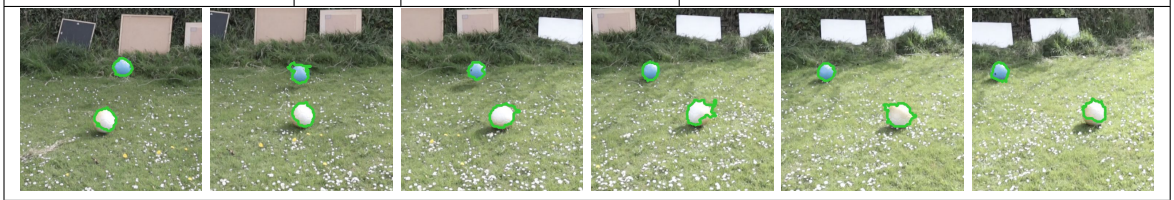
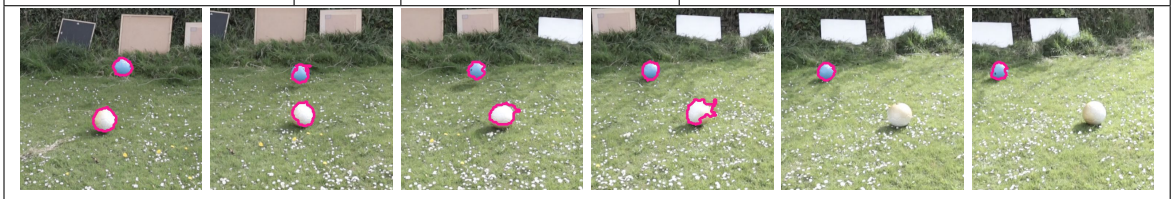
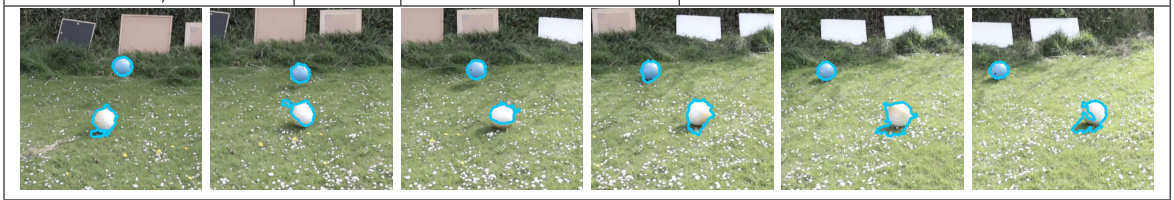
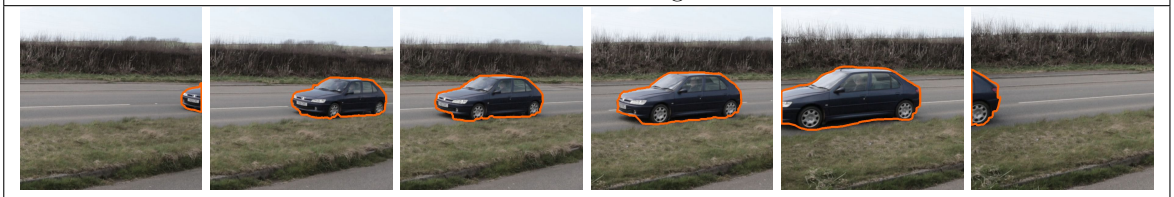
Hand-labelled Ground Truth Segmentation Mask								
								
Cropped Segmentation Results								
		Accuracy			Frames exceeding threshold			
Method	FPS	A	C_∞	$C_{0.2}$	0.6	0.7	0.8	0.9
FVS Brox flow	0.073	0.750	0.788	0.788	100.0%	77.6%	17.2%	0.0%
								
FVS basic flow	1.539	0.638	0.667	0.666	67.2%	65.5%	24.1%	0.0%
								
Ours 1280 1/2 motion	4.759	0.751	0.882	0.847	94.8%	70.7%	31.0%	1.7%
								
Ours 640 1/2 motion	19.512	0.654	0.810	0.769	69.0%	36.2%	6.9%	0.0%
								
Additional Results								
Ours 1280 1/4 motion	7.562	0.713	0.868	0.801	77.6%	50.0%	27.6%	3.4%
Ours SIFT features	1.814	0.757	0.876	0.849	96.6%	72.4%	34.5%	1.7%

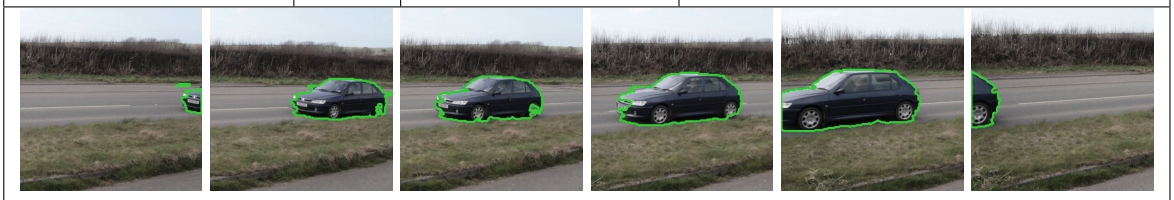
Figure 4-13: Evaluation of our method against Papazoglou's 'Fast Video Segmentation' (FVS) in a variety of configurations for the sequence 'balls'. The thumbnails show cropped segmentation results for a subset of frames, with figures computed over the entire sequence. The final two rows in the 'Additional Results' section of the table are not pictured. The accuracy criteria used is explained in Section 4.4.1.

Hand-labelled Ground Truth Segmentation Mask

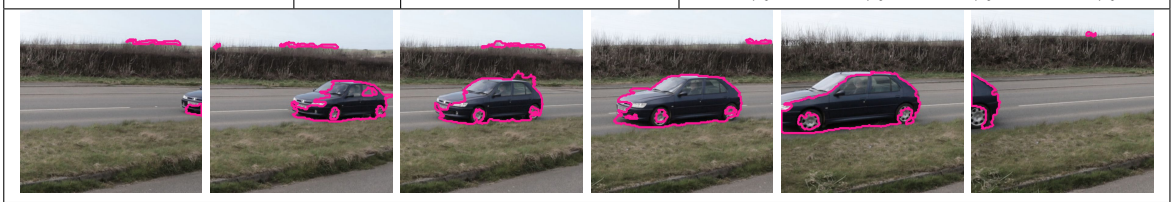


Cropped Segmentation Results

		Accuracy			Frames exceeding threshold			
Method	FPS	A	C_∞	$C_{0.2}$	0.6	0.7	0.8	0.9
FVS Brox flow	0.130	0.900	0.945	0.945	100.0%	100.0%	93.0%	62.8%

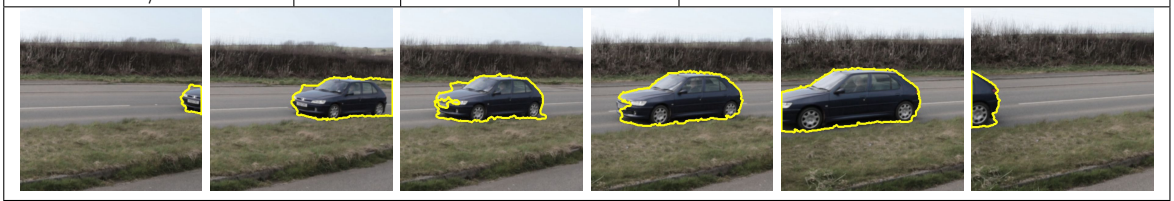


FVS basic flow	1.931	0.713	0.786	0.780	86.0%	69.8%	32.6%	0.0%



Ours 1280 1/2 motion	4.909	0.866	0.980	0.961	97.7%	95.3%	81.4%	41.9%

Ours 640 1/2 motion	19.189	0.867	0.936	0.927	97.7%	95.3%	86.0%	60.5%



Additional Results

Ours 1280 1/4 motion	7.860	0.868	0.960	0.942	95.3%	95.3%	83.7%	72.1%
Ours SIFT features	3.313	0.883	0.980	0.971	100.0%	97.7%	88.4%	51.2%

Figure 4-14: Evaluation of our method against Papazoglou's 'Fast Video Segmentation' (FVS) in a variety of configurations for the sequence 'car'. The thumbnails show cropped segmentation results for a subset of frames, with figures computed over the entire sequence. The final two rows in the 'Additional Results' section of the table are not pictured. The accuracy criteria used is explained in Section 4.4.1.

Hand-labelled Ground Truth Segmentation Mask

Cropped Segmentation Results

		Accuracy			Frames exceeding threshold			
Method	FPS	A	C_∞	$C_{0.2}$	0.6	0.7	0.8	0.9
FVS Brox flow	0.194	0.245	0.932	0.267	0.0%	0.0%	0.0%	0.0%

FVS basic flow	1.328	0.308	0.943	0.340	0.0%	0.0%	0.0%	0.0%

Ours 1280 1/2 motion	4.548	0.619	0.877	0.738	62.5%	37.5%	0.0%	0.0%

Ours 640 1/2 motion	18.927	0.208	0.262	0.243	18.8%	12.5%	0.0%	0.0%

Additional Results

Ours 1280 1/4 motion	7.323	0.323	0.404	0.383	37.5%	25.0%	0.0%	0.0%
Ours SIFT features	2.984	0.646	0.878	0.773	68.8%	50.0%	0.0%	0.0%

Figure 4-15: Evaluation of our method against Papazoglou's 'Fast Video Segmentation' (FVS) in a variety of configurations for the sequence 'horses'. The thumbnails show cropped segmentation results for a subset of frames, with figures computed over the entire sequence. The final two rows in the 'Additional Results' section of the table are not pictured. The accuracy criteria used is explained in Section 4.4.1.

Papazoglou's MATLAB implementation, where compiled MEX functions are used to perform the bulk of the computation.

We evaluated our method using a range of configurations, showing how a balance between speed and accuracy can be selected, depending on the application. Figure 4-16 shows the segmentation accuracy and total run-time (in frames per second) for several configurations of our method. These results were obtained by running each method / configuration against a dataset consisting of the four videos described in the previous section: 'balls', 'car', 'runner' and 'snowboarder'. This graph shows that, on average, our method is nearly 40 times as fast as Papazoglou for an equivalent accuracy, and even greater speed-ups in other configurations at the expense of a slight reduction in accuracy.

The run-time figure presented in these results encompasses all aspects of the algorithm; computing optical flow and confidence maps, motion correlation, superpixel segmentation, building the graph and the subsequent cut as well as the final refinement step. It is therefore representative of the actual performance, unlike in the work of [42], where run-time was presented with optical flow computation time excluded.

The fact that our method makes use of GPU processing does of course reduce the time taken to run. However, this isn't simply a case of our method being faster because we've taken the time to design a GPU implementation; rather, because of the simplicity of our method, the bulk of the computation is pixel-wise which is ideally suited to parallel processing and trivial to implement. By contrast Papazoglou's method requires global motion analysis that would be far more costly to run in a massively-parallel setting.

The results also make it apparent that our method is faster than Papazoglou's by an order of magnitude even without using sensor data. This is largely a result of our of use optical flow confidence measure that enables us to easily find regions of independent motion without being distracted by shortcomings in optical flow method, thereby enabling us to use a cheap-to-produce optical flow field. To the best of our knowledge, this use of optical flow confidence in a motion segmentation setting is novel.

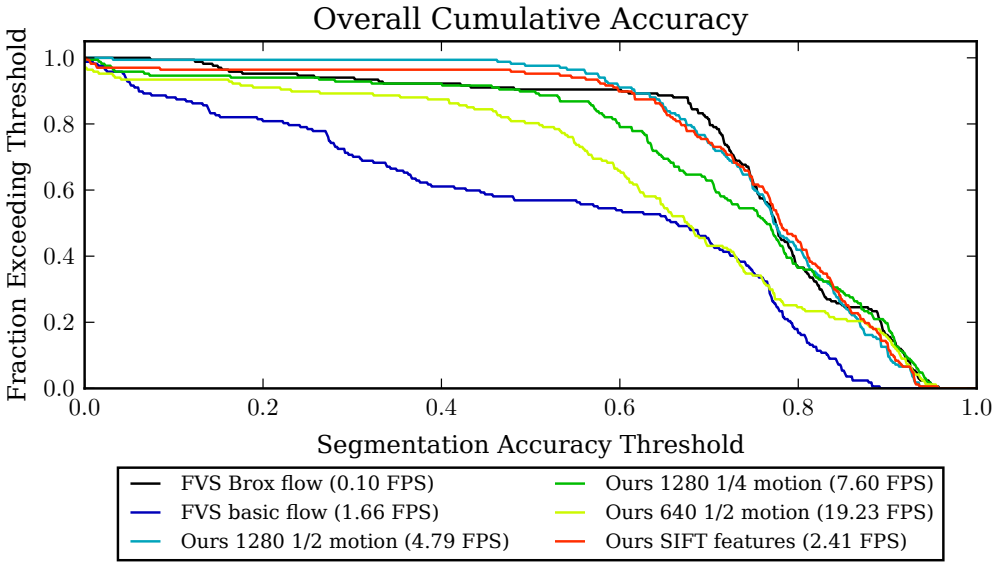


Figure 4-16: Overall run-time performance and segmentation accuracy comparison of our method in a variety of configurations, on 4 sequences with manually labelled ground truth data.

Run-time breakdowns

Here we analyse the time taken to perform the key steps of each method. Our intention is to demonstrate that the significant speed increase exhibited by our method is not merely the result of optimisation or the use of a fast optical flow scheme. This breakdown is presented in Table 4.2.

The results show that whilst optical flow computation time constitutes a significant proportion of the overall run-time in all methods, it is by far the slowest component in Papazoglou's core method, taking 10 times as long to produce as the rest of the process combined. Given this, it wouldn't be unreasonable to wonder if our speed increase was solely attributable to the fact that we use a faster optical flow method. To counter this argument we ran Papazoglou's method ('FVS basic flow') using the same optical flow algorithm and parameters as used by our method and noted that the segmentation accuracy dropped significantly as a result. This shows that their method is more reliant on having a high-quality flow field than ours, and is therefore predisposed to having a slower run-time.

Table 4.2 shows that our inertial method has very consistent run-times, with very little variation seen over the time taken to compute each element. By contrast both

Sequence	Flow	SuperPixels	Segment	Total
snowboarder	5.572 (93%)	0.090 (1%)	0.279 (4%)	5.941
balls	13.110 (95%)	0.101 (0%)	0.474 (3%)	13.685
runner	11.494 (94%)	0.097 (0%)	0.531 (4%)	12.122
car	7.248 (94%)	0.092 (1%)	0.330 (4%)	7.670
horses	4.464 (86%)	0.091 (1%)	0.599 (11%)	5.154
Average	8.378 (93%)	0.094 (1%)	0.443 (4%)	8.914

(a) FVS Brox flow

Sequence	Flow	SuperPixels	Segment	Total
snowboarder	0.026 (5%)	0.090 (17%)	0.388 (76%)	0.504
balls	0.025 (3%)	0.101 (15%)	0.524 (80%)	0.650
runner	0.025 (3%)	0.097 (13%)	0.601 (83%)	0.724
car	0.025 (4%)	0.092 (17%)	0.401 (77%)	0.518
horses	0.029 (3%)	0.091 (12%)	0.633 (84%)	0.753
Average	0.026 (4%)	0.094 (14%)	0.509 (80%)	0.630

(b) FVS basic flow

Sequence	Flow	SuperPixels	Segment	Corr	Conf	Total
snowboarder	0.061 (29%)	0.032 (15%)	0.076 (36%)	0.006 (3%)	0.033 (15%)	0.208
balls	0.061 (29%)	0.033 (15%)	0.077 (36%)	0.006 (3%)	0.033 (15%)	0.210
runner	0.061 (29%)	0.034 (16%)	0.074 (35%)	0.006 (3%)	0.033 (16%)	0.208
car	0.061 (29%)	0.032 (15%)	0.072 (35%)	0.006 (3%)	0.032 (15%)	0.204
horses	0.073 (33%)	0.032 (14%)	0.075 (34%)	0.006 (2%)	0.034 (15%)	0.220
Average	0.063 (30%)	0.033 (15%)	0.075 (35%)	0.006 (3%)	0.033 (15%)	0.210

(c) Ours 1280 1/2 motion

Sequence	Flow	SuperPixels	Segment	Corr	Conf	Total
snowboarder	0.015 (11%)	0.032 (24%)	0.070 (53%)	0.003 (2%)	0.011 (8%)	0.131
balls	0.015 (11%)	0.033 (25%)	0.070 (52%)	0.003 (2%)	0.011 (8%)	0.133
runner	0.015 (11%)	0.034 (24%)	0.073 (53%)	0.003 (2%)	0.011 (8%)	0.135
car	0.015 (11%)	0.032 (25%)	0.067 (52%)	0.003 (2%)	0.010 (8%)	0.127
horses	0.018 (13%)	0.032 (23%)	0.072 (52%)	0.003 (2%)	0.011 (7%)	0.137
Average	0.016 (11%)	0.033 (24%)	0.070 (52%)	0.003 (2%)	0.011 (8%)	0.132

(d) Ours 1280 1/4 motion

Sequence	Flow	SuperPixels	Segment	Corr	Conf	Total
snowboarder	0.015 (28%)	0.008 (15%)	0.016 (30%)	0.003 (5%)	0.010 (19%)	0.052
balls	0.015 (28%)	0.009 (16%)	0.015 (29%)	0.003 (5%)	0.010 (19%)	0.052
runner	0.015 (28%)	0.009 (16%)	0.016 (30%)	0.003 (5%)	0.010 (19%)	0.052
car	0.015 (29%)	0.008 (15%)	0.016 (30%)	0.003 (5%)	0.010 (19%)	0.052
horses	0.018 (33%)	0.008 (15%)	0.014 (26%)	0.003 (5%)	0.010 (18%)	0.053
Average	0.016 (29%)	0.008 (16%)	0.015 (29%)	0.003 (5%)	0.010 (19%)	0.052

(e) Ours 640 1/2 motion

Sequence	Flow	SuperPixels	Segment	Corr	Conf	SIFT/Fit	Total
snowboarder	0.061 (18%)	0.032 (10%)	0.096 (30%)	0.006 (1%)	0.032 (9%)	0.092 (28%)	0.319
balls	0.061 (11%)	0.033 (6%)	0.069 (12%)	0.006 (1%)	0.034 (6%)	0.348 (63%)	0.551
runner	0.061 (12%)	0.034 (6%)	0.073 (14%)	0.006 (1%)	0.032 (6%)	0.298 (59%)	0.505
car	0.060 (20%)	0.032 (10%)	0.066 (21%)	0.006 (2%)	0.032 (10%)	0.104 (34%)	0.302
horses	0.072 (21%)	0.032 (9%)	0.080 (23%)	0.006 (1%)	0.032 (9%)	0.113 (33%)	0.335
Average	0.063 (15%)	0.033 (8%)	0.077 (19%)	0.006 (1%)	0.032 (8%)	0.191 (47%)	0.402

(f) Ours SIFT features

Table 4.2: Performance breakdowns by method / configuration and sequence, with time taken in seconds for each core component. 'Flow' refers to optical flow computation. 'SuperPixels' is segmenting the image into superpixels. 'Segment' is the overarching process of analysing the motion, building the graph and cutting it, and any refinement steps. 'Corr' refers to finding motion correlation (e.g Figure 4-3a). 'Conf' is optical flow confidence (e.g Figure 4-3b). 'SIFT/Fit' corresponds to finding SIFT feature correspondences, performing RANSAC filtering and fitting a transform. The percentage in brackets refers to the proportion of the total time for the frame taken to compute each aspect.

Papazoglou's and ours with fitted motion show significant disparity from sequence to sequence. This facet is particularly important when considering real-time applications, where uniform computation times ensure reliable frame-rates and lessen hardware requirements as the worst-case scenario is very close to the average.

4.4.3 Sensor Benefit

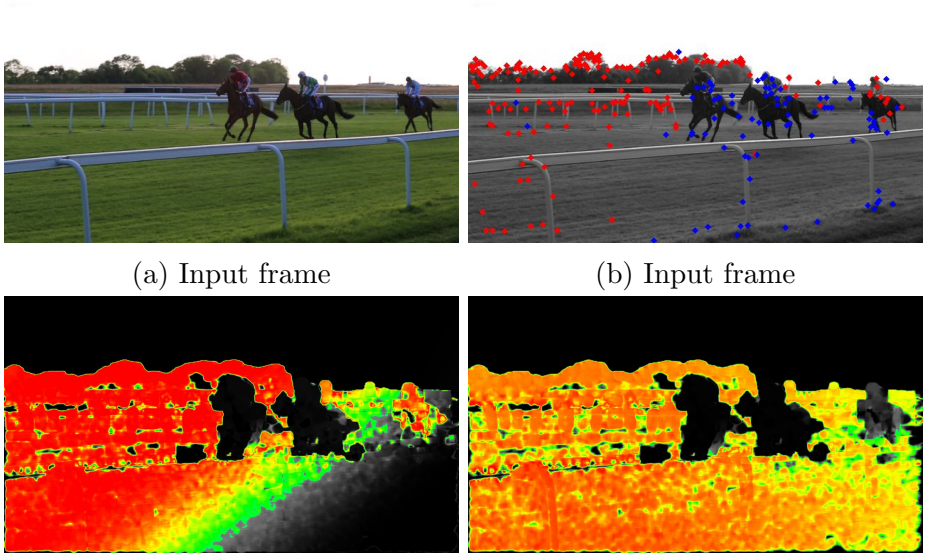
The performance in terms of segmentation accuracy of our method when using either inertial sensor data or motion inferred from sparse feature correspondences is very similar. This is not an unexpected result; assuming there are enough features distributed across the image for the feature detector to find, and enough of these are static such that the RANSAC step keeps only static points, then the motion found from these correspondences is likely to be very accurate.

This result perhaps reinforces the value of the inertial sensor data, as we are able to use it to achieve a very similar result without requiring the costly process of finding feature correspondences and fitting an affine transform, a step where performance is not only slow but also variable depending on the scene. This is apparent in Table 4.2 where the 'SIFT/Fit' step takes more than 3 times as long on average for the 'balls' scene as it does for 'snowboarder'. On average our method using inertial sensor data is more than twice as fast as the fitted case for an equivalent segmentation accuracy.

It's also important to consider that the feature identification and matching functions of OpenCV consist of highly optimised compiled code and yet take longer to produce a result than the rest of our experimental segmentation implementation combined.

Feature identification and matching is a complex, high-level task. We've shown that sensor data offers an accurate alternative that can be computed in a fraction of the time (when comparing the relative times of fitting motion and integrating inertial sensor data). For real-time applications, where a hardware implementation of motion segmentation might be of great benefit, the complexity of implementing the motion fitting step is both costly from the perspective of development and silicon area, a problem completely avoided by using inertial sensor data.

No discussion on the merits of sensor data against fitted motion would be complete without also discussing the failure modes. Fitting motion to sparse features from an image is very likely to result in a flow field that accurately matches the motion of at least some portion of the image, but there is also the chance that it



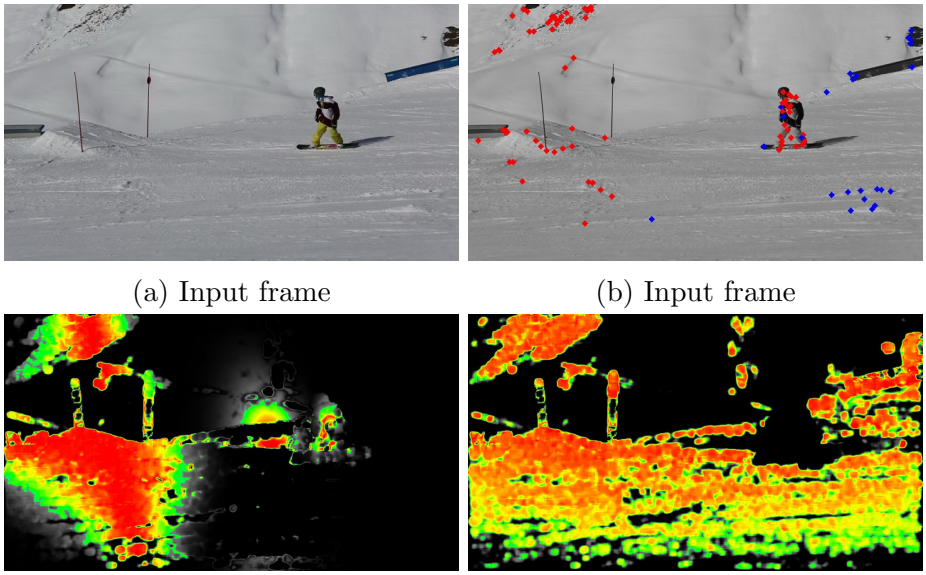
(a) Input frame (b) Input frame

(c) Motion fitted to sparse features (d) Motion according to sensors

Figure 4-17: Mis-fitting motion. (a) shows the input frame. In (b) the features have been marked, with the blue ones removed by the RANSAC filtering step leaving just the red to be fitted. (c) and (d) shows the motion correlation of the fitted motion and sensor motion respectively. In the fitted case we can clearly see the result of the poor fit, with no correlation in the lower right of the image.

either corresponds to scene motion or badly fits the rest of the image.

A good example of such a failure can be seen in Figures 4-17 and 4-18. These results were found using OpenCV's findHomography function which takes pairs of feature correspondences from adjacent frames as input and finds a transformation **H** describing the best fitting motion. RANSAC is used to find the set of inliers / outliers and corresponding transformation which minimises the back-projection error. It is this latter step that is susceptible to fitting the wrong motion. In these examples the features are not evenly distributed throughout the image but instead are clustered together. This results in a transformation that, whilst fitting the majority of features well, fails to accurately describe the true motion across the entire image. By contrast the sensor data gives a better overall measure of the motion, even if the maximum is not as high as the fitted motion, with low correlation for the moving areas in each example.



(a) Input frame (b) Input frame

(c) Motion fitted to sparse features (d) Motion according to sensors

Figure 4-18: An additional example of badly fitted motion to sparse features. Refer to figure 4-17 for a description of the images. The features used for fitting the transform, marked in red, correspond to only a small section of the image and so the motion does not fully represent actual movement of the camera.

4.5 Conclusion

In this chapter we've described a real-time method for identifying and accurately segmenting independently moving non-rigid objects in images filmed by a moving camera using inertial sensor data. We have provided quantitative analysis that shows that our simple method is much faster, less restrictive (using only a single frame pair) and more consistent than the current state-of-the-art in fast motion segmentation methods by at least an order of magnitude, whilst achieving equivalent segmentation accuracy. This improvement is significant because it now makes real-time motion segmentation a reality, opening up new applications.

We have added weight to our hypothesis by evaluating the state of the art method for fast video segmentation [42] with both expensive, high-quality and cheap, low-quality optical flow fields, and shown the reduction in matching performance in the latter case. This is key to the power of our method of combining the inertial sensor data with optical flow confidence; we are able to easily make sense of the noisy flow data because of our prior knowledge. We also demonstrated the utility of sensor data by comparing it to camera motion found from SIFT feature correspondences,

yielding very similar segmentation accuracy at a much lower computational cost.

In the overall context of this thesis, we've shown an ideal application of inertial sensor data. Whilst we've shown that inferring camera motion from sparse feature correspondences can, in many cases, produce equivalent results in terms of accuracy, we've also demonstrated various shortcomings; speed, variable run-time, failure modes and additional complexity. This is where the benefit of sensor data really shines through, as it enables a method that is not only faster but is also simpler and therefore cheaper to implement without the reliability concerns, factors that are crucial when considering real-time applications.

4.5.1 Future Work

In this work we've restricted our camera motion to rotation. Extension to translational motion would be a logical next step. We've highlighted the difficulties of accurately tracking camera translation in Chapter 2, but in this application we have the advantage of using the images, specifically the static regions, as feedback to the motion estimation. The Extended Kalman Filter (EKF) would be a good choice for fusing the noisy data from the inertial sensors with feedback from the images to update an estimate of the system state, comprising position, orientation and velocity of the camera.

Further performance increases could be found by exploiting the inertial sensor data when computing motion correlation. The inertial data could be used as part of the optical flow step as an initial solution, which would allow for a constrained search space. Our method is not reliant on a full optical flow field; it would be sufficient to just identify areas where the image motion is not as predicted by the sensor data.

Chapter 5

Understanding Motion Blur

Motion blur is a common problem in photography. If the image focused on the camera sensor moves during exposure the response will be blurred. The extent of the blur, which is not always unintentional, depends on the magnitude of motion and the length of exposure. In many cases the extent of the motion will be small enough that the blur is imperceivable. Motion blur can be avoided by either minimising the relative motion (keeping the camera still, reducing the scene motion or following the motion with an equivalent camera motion) or by reducing the exposure time. There comes a point where motion blur becomes unavoidable, whether as a result of not being able to reduce motion or when the exposure time is at the limit due to the technical capabilities of the image sensor.

Recovering the original unblurred image from a captured blurry image is not a straightforward task. Even when making a static scene assumption, such that all blur is the result of motion of the camera, there is still a very large search space of likely camera motions. When coupled with an unknown scene, possibly containing motion, the problem of deblurring an image becomes intractable. However, past works have shown that a combination of a model relating the desired unblurred image to the observed blurry image, together with a priori assumptions about the content of an image and the form of the blur, can recover a sharpened image in many situations.

In this chapter we discuss the causes and effects of motion blur, as well as the process of reversing the blur. We relate this work to the information provided by inertial sensors and show how they can be used to better understand motion blur. We describe our experiments in which we use the sensor data to detect scene motion

in a blurry image.

It is important to note that image blur can be caused by several factors in addition to motion, such as an unfocused lens or optical imperfections. Our focus is primarily on blur caused by motion, and so we assume that if otherwise stationary, an image would have contained sharp edges whose recovery forms the target for deblurring. We assume that the optical properties of the camera, particularly its focal length, remain fixed throughout the exposure.

5.1 Background

In this section we formulate a mathematical model of motion blur, discuss common techniques used to speed up the computational side of image deblurring and blur estimation, and introduce the problem of scene motion blur.

5.1.1 Blur Model

A moving camera of known intrinsics, $\mathbf{K}$ (see Chapter 2), captures a blurry image $\mathbf{B} \in \mathbb{R}^N$. The unblurred latent image, $\mathbf{L} \in \mathbb{R}^N$, is chosen as one the instantaneous images that would have been projected on the image sensor at some point throughout the exposure. We opt for $\mathbf{L}$ to be the image seen when the camera is positioned and orientated at the centre point of of the range of motions it experiences throughout the exposure. Making this distinction is entirely arbitrary as in reality there is an infinite number of latent images evenly distributed throughout the exposure.

The blurry image, captured by an image sensor comprised of a grid of discrete pixels, is the integration of the light intensities projected on to each pixel. The blurred image is then defined as:

$$B(x) = \int \Omega(x) + n \tag{5.1}$$

where $\Omega(x)$ is the light incident on the image sensor plane at location x, and n is a random noise element. A Gaussian distribution is commonly used to model noise in image processing, with n being a Gaussian random variable with mean μ and variance σ^2, written as $n \sim \mathcal{N}(\mu, \sigma^2)$.

The blurred image can be approximated as the weighted average of a series of discrete sharp images. If we consider $\mathbf{V}$ as a set of M images of the scene that the

camera may have seen at some point throughout the exposure, with each row in $\mathbf{V}$ being the result of an affine transformation applied to $\mathbf{L}$, then we can think of $\mathbf{B}$ as being the weighted sum of the images in $\mathbf{V} \in \mathbb{R}^{N \text{x} M}$:

$$\mathbf{B} = \mathbf{V}\mathbf{A} + n \tag{5.2}$$

where $\mathbf{A} \in \mathbb{R}^M$ is a vector whose sum is 1, with each element corresponding to the proportion of time, A_i, the camera spent viewing the corresponding image $\mathbf{V}_i \in \mathbb{R}^N$. Alternatively, we can consider $\mathbf{L} \in \mathbb{R}^N$ to be the unblurred latent scene image, and $\mathbf{X} \in \mathbb{R}^{N \text{x} N}$ to be a transformation matrix that applies a blur to $\mathbf{L}$:

$$\mathbf{B} = \mathbf{X}\mathbf{L} + n \tag{5.3}$$

The use of these two distinct forms varies depending on which of $\mathbf{A}$ or $\mathbf{L}$ is being estimated. A standard practice in image deblurring is to alternatively estimate one of these two quantities whilst the other remains fixed. When not referring to a specific form we use $*$ to denote the process of blurring $\mathbf{L}$ according to the camera motion $\mathbf{A}$:

$$\mathbf{B} = \mathbf{L} * \mathbf{A} + n. \tag{5.4}$$

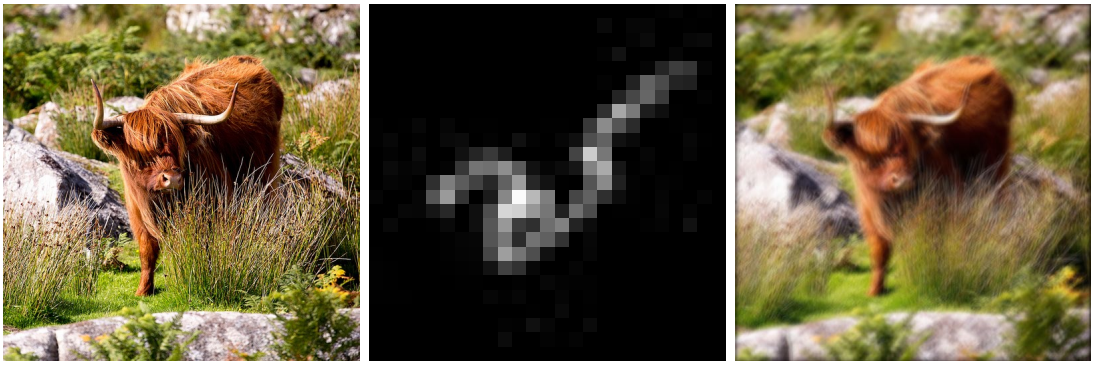
Frequency Domain Analysis

Point Spread Functions (PSFs) are a convenient representation of a blur function, and in this context are often referred to as 'blur kernels'. These kernels describe how the intensity of each pixel in $\mathbf{L}$ is distributed amongst its neighbours in $\mathbf{B}$. The blur kernel can be applied to an image using convolution:

$$B(t) = \int_{-\infty}^{\infty} L(\tau)k(t-\tau) \cdot d\tau \tag{5.5}$$

where $\mathbf{k} \in [0, 1]$ is a 2D blur kernel whose elements sum to 1. Figure 5-1 shows the result of applying a blur kernel to an image.

Convolution and deconvolution can be performed very quickly in the frequency domain as a multiplication or division respectively, a property exploited in many debluring methods. Equation 5.6 shows how $\mathbf{L}$ can be blurred by a kernel $\mathbf{k}$, after



(a) Image (b) Kernel (c) Blurred Image

Figure 5-1: The latent image in (a) is blurred using the blur kernel in (b), giving the blurry result in (c).

appropriate padding and shifting, to produce a blurred image $\mathbf{B}$:

$$\mathbf{B} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{L}) \circ \mathcal{F}(\mathbf{k})) \tag{5.6}$$

where $\mathcal{F}(\mathbf{x})$ and $\mathcal{F}^{-1}(\mathbf{x})$ are the discrete 2D Fourier and inverse Fourier transforms of $\mathbf{x}$ respectively.

Image analysis in the frequency domain is not without its limitations. Image data at the boundaries will cause boundary artefacts that can be compensated for, typically with padding and a window function that gradually reduces to zero at the extremity. Figure 5-2 shows an example of *ringing*, which is a corruption of the image as a result of noise, most commonly found around sharp edges.

Camera Motions

The motion of a camera can be expressed with 6 degrees of freedom (DOF); translation in x, y, z and rotation x, y, z. Modelling the motion in this way allows for the full range of motion-induced blurs to be described, but comes at the cost of requiring a huge search-space when searching for the optimal camera motion. It is for this reason that assuming camera motion to be constrained within the plane of the image sensor is convenient, allowing the blur function to be assumed uniform over the entire image. The assumption of spatially invariant blur formed the basis of pioneering deblurring works such as [48].

Spatially invariant motion models are limited in the motions they can describe,

(a) $\mathbf{B} = \mathbf{L} * \mathbf{k} + n$ (b) $\mathcal{F}^{-1}(\frac{\mathcal{F}(\mathbf{B})}{\mathcal{F}(\mathbf{k})})$

Figure 5-2: Artefacts arising from deconvolution in the frequency domain. (a) is the result of blurring the image in Figure 5-1a with the kernel in Figure 5-1b and then adding Gaussian noise $\mathbf{n} \sim \mathcal{N}(0, 1 \times 10^{-5})$. (b) is the result of division in the frequency domain with the same kernel to get the deconvolved result. Severe ringing artefacts can be seen throughout the image, particularly at the edges.

and are easily invalidated with real images, whilst 6DOF models are impractical to implement. A compromise is to use an approximate 3DOF model that allows commonly occurring blurs to be expressed whilst not requiring such a large search space. Translation of the camera towards the scene can be assumed to be negligible, both due to its small impact (assuming a reasonable distance from camera to subject) and the fact that it is less likely to occur in accidental blur scenarios than other camera motions. Translation along the x axis produces very similar motions to rotations about y, with any differences decreasing with increased focal length. These observations have been exploited for spatially varying blur models that can handle a wide-range of commonly occurring camera motions, whilst assuming camera motion to be either purely rotational: R_x, R_y, R_z (*e.g.* [49]) or translational in the image plane with rotations about the optical axis: T_x, T_y, R_z (*e.g.* [50, 51]).

Spatially-varying motion blur functions make frequency domain computation across the entire image unfeasible as there is not one single PSF that describes the blur at every pixel. Hirsch *et al.* [50] work around this limitation by assuming spatially invariant blur within small overlapping windows which are then blended together. This approach has later been used to speed up spatially-varying image deblurring [52, 49].

Camera Pose Space

The camera pose space defines the search space of all possible motions the camera may have exhibited whilst capturing the blurred image. Each discrete pose in this space represents the image seen by the camera at a particular position and orientation.

It is desirable to reduce the size of this search space to be no larger than necessary whilst still being able to express the motions that caused the blur. The problem with this is that the possible ranges of camera motion are not known in advance, so camera motion magnitude is usually a parameter that is left to the user to set.

Hu and Yang [53] show that the time taken to deblur an image can be decreased by limiting the number of camera poses to consider. They achieve this by using a fast spatially invariant method to deblur several small patches of the blurred image and then fit a spatially varying camera motion to the kernels. When they come to estimating camera motion over the full image, they use a pose space containing only those poses found from the patches.

5.1.2 Probabilistic formulation

We can define a formal probabilistic model for blind image deblurring using Bayes' theorem:

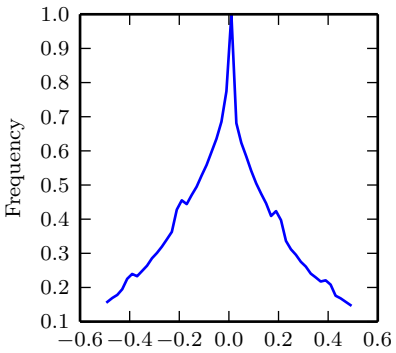
$$p(\mathbf{L}, \mathbf{A}|\mathbf{B}) \propto p(\mathbf{B}|\mathbf{L}, \mathbf{A})p(\mathbf{L})p(\mathbf{A}) \tag{5.7}$$

where the prior terms $p(\mathbf{L})$ and $p(\mathbf{A})$ allow us to apply prior knowledge to the unknowns. The problem of image deblurring can then become one of maximising $p(\mathbf{L}, \mathbf{A}|\mathbf{B})$; that is, finding the sharp image $\mathbf{L}$ and camera motion $\mathbf{A}$ that best explains the blurred observation $\mathbf{B}$, given our prior assumptions. This process is known as 'maximum a-posteriori'.

5.1.3 Deblurring Priors

Restoring the unblurred image is an inherently ill-posed problem. It can be made more tractable by use of prior knowledge to draw the solution towards desirable results and avoid the trivial solution, where $\mathbf{L} = \mathbf{B}$ and $\mathbf{A}$ represents a static camera.

(a) 'Natural' image



(b) Gradient Distribution

Figure 5-3: Gradient distribution for a 'natural' image. (b) shows how small gradients occur much more frequently than strong edges

Camera Motion

The camera motion priors guide the solution away from unlikely motions. It is reasonable to assume that the camera has not moved uniformly throughout the entire pose space, so a sparse motion representation should be preferred. The motion of the camera should not be disjoint, so it is advantageous to minimise the gradients in $\mathbf{A}$. The actual form of this prior varies, for example, Shan *et al.* [54] regularise $\mathbf{A}$ using the L1-norm, whereas in [55, 56] the L2-norm is used.

Sharp Image

Priors on the unblurred image encourage the solution to contain sharp edges. The gradient distributions in natural images, which are real images of the world and its contents, can be seen to exhibit a heavy-tailed distribution in which a small proportion of pixels have a high gradient magnitude whilst the vast majority have a low gradient magnitude. Figure 5-3 shows the gradient distribution for an example image.

Priors of this type have been used extensively in deblurring literature, for example in [48, 54, 57]. Such a prior leads to a non-convex objective function, so the distribution is approximated with a function more suited to optimisation, of which there are several forms.

5.1.4 Edge Mask

Latent image refinement involves searching for a deconvoluted image that satisfies both estimated camera motion and the sharp image priors. Owing to the fact that sharp edges are specifically sought, there is the possibility that such features are 'halucinated' where the edge did not exist in reality, perhaps as a result of ringing artefacts from frequency space analysis. For this reason it is not desirable to apply the sharp edge prior for all areas of an image. Joshi *et al.* [58] limit sharp edge estimation during deconvolution to pixels near to predicted edges. Shan *et al.* [54] compute the standard deviation of pixel intensities in local windows. Where this value is less than a threshold, they constrain the blurred image gradients to be similar to the unblurred image gradients.

The success of kernel estimation is largely dependent on the image structure and edges being suitable for motion estimation. The suitability of an edge can vary depending on its size and direction relative to the blur function. Xu and Jia [56] describe the 'r-map' that weights the edges used in kernel estimation with a metric that penalises structures smaller than the blur kernel.

5.1.5 Computation

The mathematical operations required to perform image deconvolution and blur kernel estimation are very costly as they require repeated evaluation of many millions of pixel values, and estimation of millions of unknowns when producing the unblurred image. Techniques for deblurring images have evolved with these considerations in mind, and whilst not always optimal in formulation, are compromises based on available processing power and memory.

To be able to understand the complications that can arise in image deconvolution and blur motion estimation, we describe the general underlying form of many recent deblurring works and highlight the costly operations required.

Regression plays an important role in both image deconvolution and blur kernel estimation. The model in Equation 5.2 can be re-arranged to an energy minimisation problem, where the target is to minimise the differences between the observed blurry image and the estimated latent image when blurred by the estimated blur function:

$$E(\mathbf{L}, \mathbf{A}) = \|\mathbf{L} * \mathbf{A} - \mathbf{B}\|^2 + \boldsymbol{\theta} \tag{5.8}$$

where $\boldsymbol{\theta}$ is some form of regularisation for the priors, typically on both the image and the camera motion.

During camera motion estimation, the latent image $\mathbf{L}$ is fixed whilst the estimate of camera motion $\mathbf{A}$ is refined. Ignoring the priors for now, the energy function then reduces to:

$$E(\mathbf{A}) = \|\mathbf{V}\mathbf{A} - \mathbf{B}\|^2 \tag{5.9}$$

which can then be differentiated with respect to $\mathbf{A}$ with some rearrangement and careful handling of the matrices:

$$\frac{\partial E(\mathbf{A})}{\partial \mathbf{A}} = 2\mathbf{V}^{\mathrm{T}}\mathbf{V}\mathbf{A} - 2\mathbf{V}^{\mathrm{T}}\mathbf{B} \tag{5.10}$$

Direct computation of Equation 5.10 is non-trivial due to the enormous size of $\mathbf{V}$. However, this process can be achieved relatively quickly in the frequency domain:

$$\mathbf{V}^{\mathrm{T}}\mathbf{V}\mathbf{A} = \mathcal{F}^{-1}(\overline{\mathcal{F}(\mathbf{L})} \circ \mathcal{F}(\mathbf{L}) \circ \mathcal{F}(\mathbf{A})) \tag{5.11}$$

where $\overline{\mathbf{x}}$ denotes the complex conjugate of $\mathbf{x}$. $\mathbf{V}^{\mathrm{T}}\mathbf{B}$ is similarly found:

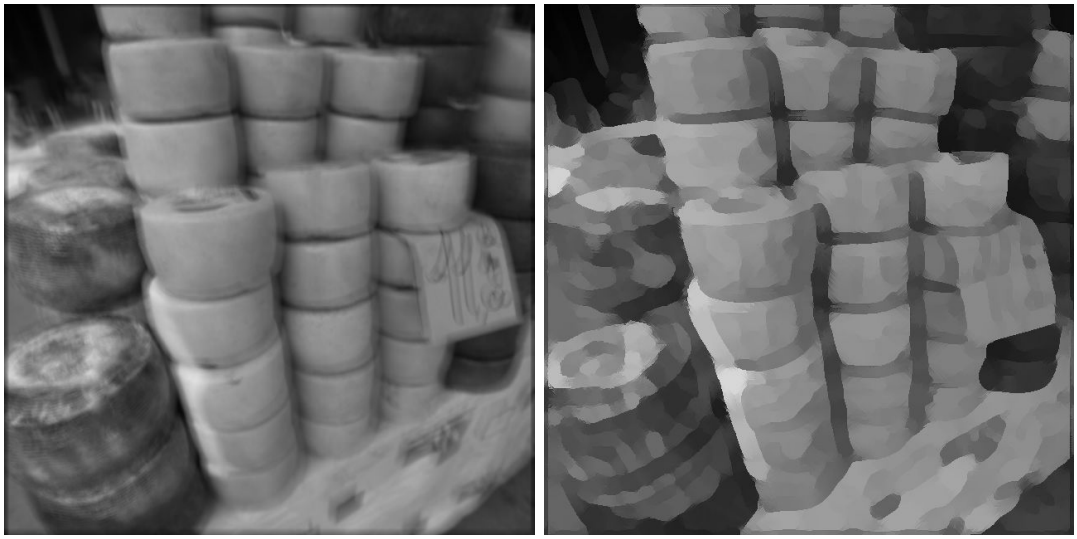
$$\mathbf{V}^{\mathrm{T}}\mathbf{B} = \mathcal{F}^{-1}(\overline{\mathcal{F}(\mathbf{L})} \circ \mathcal{F}(\mathbf{B})) \tag{5.12}$$

Performing these operations within the frequency domain has its associated problems, as we've already mentioned. Extensions to spatially varying blur, such as [52, 59], use overlapping windows with a similar form to equation 5.11. The patch-wise windows are computed individually and then fit to the global motion space.

The forms given up until this point operate directly on the pixel intensities. This is not ideal as the periodicity characteristics of discrete FFTs will result in boundary artefacts. This problem can be avoided by instead operating on image gradients. The deblurring energy function then becomes one that seeks to minimise the difference between the gradients in the blurred image and the gradients in the unblurred image when blurred according to the blur function:

$$E(\mathbf{L}, \mathbf{A}) = \sum_{\partial^* \in \boldsymbol{\Theta}} \|\partial^*\mathbf{L} * \mathbf{A} - \partial^*\mathbf{B}\|^2 + \boldsymbol{\theta} \tag{5.13}$$

where ∂^* is a partial derivative operator from the set of all operators $\boldsymbol{\Theta}$. The use of image gradients in image deblurring can be seen in [60, 55, 56, 52, 61, 53, 51]



(a) Blurred Image (b) Sharp Prediction

Figure 5-4: Sharp Image Prediction

amongst others. A common form is the one proposed by Shan *et al.* [54], who use a weighted set of zero (original intensities), first and second order gradient operators in x, y and diagonally in xy.

5.1.6 Blind Deblurring Process

In blind image deblurring we have no knowledge of the content of the unblurred scene or the camera motion, just a blurred image $\mathbf{B}$. Therefore we are limited to making assumptions about the likely content of $\mathbf{L}$ and a preference towards certain types of camera motion.

This is very much a chicken and egg problem; given $\mathbf{L}$ and $\mathbf{B}$, we can estimate the camera motion using the method previously stated. Or, if we already know the camera motion, we can use a non-blind deconvolution process to find $\mathbf{L}$. A popular technique for solving this dilemma is to make a prediction of the sharp edges in $\mathbf{L}$ using the shock filter of Osher and Rudin [62]. The shock filter is evolved as:

$$\mathbf{I}_{t+1} = \mathbf{I}_t - \text{sign}(\Delta \mathbf{I}_t) \|\nabla \mathbf{I}_t\| dt \tag{5.14}$$

where dt is the timestep, $\mathbf{I}_t$ is the image at time t, and $\Delta \mathbf{I}_t$ and $\nabla \mathbf{I}_t$ represent the laplacian and gradient of the image respectively. A bilateral filter is used as a pre-

processing step to remove noise in flat regions prior to applying the shock filter. $\hat{\mathbf{L}}$ is then the sharp prediction, computed on the current estimate of $\mathbf{K}$. The sharp prediction for an example image is shown in Figure 5-4.

Input: Blurry image $\mathbf{B}$
Output: Unblurred image $\mathbf{L}$, Camera motion $\mathbf{A}$
Initialisation:
$\mathbf{L}_{t=0} = \mathbf{B}$
$\boldsymbol{\mathcal{B}}$ is the set of $\mathbf{B}$ at different scales
for $\hat{\mathbf{B}} \in \boldsymbol{\mathcal{B}}$ **do**
  repeat
    $\hat{\mathbf{L}} = \text{sharpPrediction}(\mathbf{L}_t)$
    $\mathbf{A}_{t+1} = \text{estimateCameraMotion}(\mathbf{B}, \hat{\mathbf{L}})$
    $\mathbf{L}_{t+1} = \text{deconvolve}(\mathbf{B}, \mathbf{A}_{t+1})$
  until $\|\mathbf{L}_t - \mathbf{L}_{t-1}\| < \tau_L$ *and* $\|\mathbf{A}_t - \mathbf{A}_{t-1}\| < \tau_A$;
end

Algorithm 3: Generalised Blind Motion Deblurring Process. The outer loop starts with a small scale image and works up to the full-size $\mathbf{B}$. The inner-most loop is repeated until convergence.

The process in Algorithm 3 outlines a generic image-deblurring method. A common practice in image deblurring is to rely more heavily on prior assumptions and only the most salient edges whilst the estimate of $\mathbf{L}$ is in the early stages. This is achieved in several ways. Firstly, the use of an image pyramid ensures that the image is blurred according to the motion of the most prominent features first. Secondly, the weighting of the priors against the data term is reduced as time goes on, so the result relies less on prior assumptions and more on the content of the image. Additionally, the parameters used to estimate the sharp prediction $\hat{\mathbf{L}}$ can change as time progresses, with the effect of the bilateral reduced, causing weaker edges to be included.

Our experience of implementing previous works such as [50] has shown the deblurring result to be extremely sensitive to the evolution of these weights. In the few published deblur implementations it is not uncommon for each image to have its own specific parameters.

5.1.7 Deblurring difficulties

Deblurring is an inherently difficult problem and is affected by many issues, as was evident recently when Adobe demonstrated some new deblurring technology in their Photoshop product. They used images that were later shown[1] to be synthetically blurred, ensuring the blur function met their assumptions. Whilst the result was still impressive and it is good to see these technologies make their way into commercial products, this highlights the difficulties involved with practical image deblurring, and the need to use source images that fit the assumptions. We now go on to discuss a few common problems that can arise in image deblurring.

Variable Depth

A severe restriction of current deblurring technology is its ability to handle images with a variety of depths. Unfortunately such images are commonplace in reality, and the assumption of planar content is easily violated. A scene with varying depths causes the blur function to vary non-linearly across the image. Xu and Jia [60] tackle this problem by inferring the depth from a blurred stereo image pair and estimate the blur function for many small regions.

Pixel Saturation

An image that contains areas that are much brighter than the rest of the image will likely be exposed according to the darker parts. The result of this is that the pixel values of the bright areas will be overexposed, leading to saturated pixels. The models described so far assume the blurred image to be a linear combination of unblurred images, but in the case of saturated pixels this assumption does not hold.

Whyte *et al.* [59] experiment with both excluding pixels whose intensity exceeds a threshold from kernel estimation and by modifying their model to take into account the non-linear response. Cho *et al.* [63] propose a blur model that takes these outliers into consideration, and show it can reduce artefacts in deconvolution.

[1] http://blogs.adobe.com/photoshopdotcom/2011/10/behind-all-the-buzz-deblur-sneak-peek.html

Scene Motion

An object that moves independently of the rest of the scene will have a blur function that differs from the static elements. This disrupts the kernel estimation and can cause severe artefacts in the deconvolution step as it searches for sharp edges that would not be found in the moving regions. The impact of scene motion on image deblurring is considered in more detail for the remainder of this chapter.

5.2 Understanding Image Blur using Inertial Sensor Data

The data from the inertial sensors gives us an estimate of how the camera moved whilst capturing a blurry image. Using the methods described in Chapter 2, we find the position and orientation of the camera at each discrete time-step. We convert these estimates to be relative to a central pose, such that the average translation and rotation is $[0, 0, 0]^T$. Using these estimates, we can build a set of homography matrices corresponding to each pose in the motion space the camera visited. We define this set $\boldsymbol{\mathcal{H}}$, with each element $\boldsymbol{\mathcal{H}}_i \in \mathbb{R}^{3\text{x}3}$. The blurred image can then be estimated according to the sensor data:

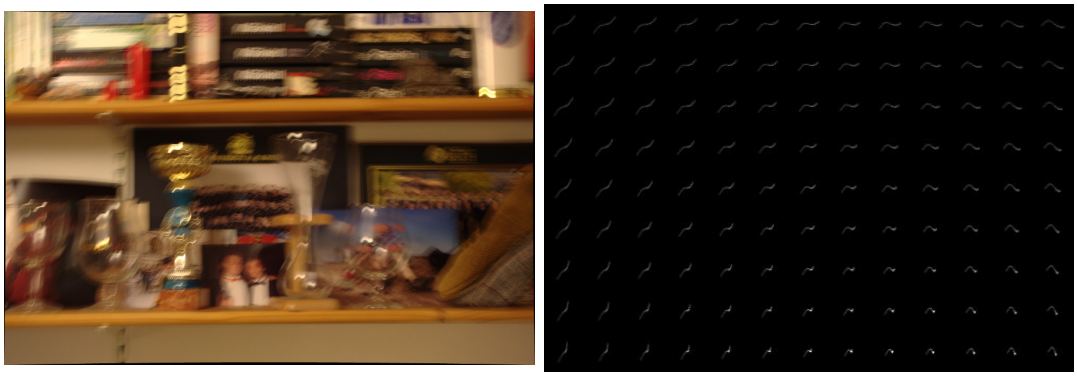
$$\mathbf{B} = \sum_{\mathbf{H} \in \boldsymbol{\mathcal{H}}} \beta(\mathbf{H}, \mathbf{L}) \tag{5.15}$$

where $\beta(\mathbf{H}, \mathbf{L})$ is a function that transforms $\mathbf{L}$ according to the affine homography $\mathbf{H}$.

The blur kernel at a point $\mathbf{p} = [x, y, 1]^{\mathrm{T}}$ in the image is found as a linear combination of an impulse transformed by each $\boldsymbol{\mathcal{H}}_i$:

$$k(\mathbf{p}) = \sum_{\mathbf{H} \in \boldsymbol{\mathcal{H}}} \alpha(\mathbf{H}\mathbf{p} - \mathbf{p}) \tag{5.16}$$

where $\mathbf{H}\mathbf{p} - \mathbf{p}$ is the relative change in position of $\mathbf{p}$ according to the motion $\mathbf{H}$ (with appropriate treatment of the homogeneous coordinate), and $\alpha(\mathbf{q})$ is a function that shifts an image of an impulse centred at $(0, 0)$ by $\mathbf{q}$ pixels in the x and y directions. It is likely that the shift will be to a sub-pixel location, so bilinear interpolation is used.



(a) Blurred Image (b) Kernel Prediction

Figure 5-5: A blurry image and the predicted blur functions according to the inertial sensors. The true form of the blur is evident in the various bright highlights.

The spatially varying blur function can be visualised as a grid of blur kernels at various points across the image. This is produced by finding $k(\mathbf{p})$ for the pixel at the centre of each patch. Figure 5-5 shows the sensor-predicted blur kernels for a captured image.

Scene motion causes regions of the image to be blurred independently of the background, which invalidates the blur models we've discussed. Our focus is on blurred images that, with the exception of one or more moving elements, meet the assumptions of the models discussed previously.

The effects of scene-motion are two-fold. Firstly, it impacts camera-shake estimation, causing the estimated motion to be influenced by the scene motion. Secondly, it will cause deconvolution artefacts where sharp edge priors are applied to moving regions. In each case the problem can be mitigated if the locations of the moving elements are known, by estimating camera motion from only static points and only applying sharp edge priors to static regions. In light of this, we focused our efforts on the segmentation of moving objects from blurry images.

In contrast to other aspects of image deblurring, scene motion has received relatively little attention. To date there have been several works that segment scene motion in a blurry image captured by a static camera. Levin *et al.* [64] use the statistics of gradient distributions computed in local windows to identify regions with differing motions. Chakrabarti *et al.* [65] develop a local blur cue that rates the likelihood that a candidate blur kernel caused the observed blur, and use it to segment motion blur, exploiting the assumption that similar colour regions exhibit

similar motions. Bahrami *et al.* [66] segment an image based on out-of-focus and motion blur, using local blur kernels to group regions. None of these methods use images that are corrupted by both scene and camera motion.

There are methods for analysing motion in blurred images, such as those in the work by Cho *et al.* [67], that utilise several images to infer the moving regions. Our motivation is to produce results that require just one blurred image and the sensor data.

In contrast to these works, our preliminary investigations show that inertial sensors can be used to isolate independent scene motion from a single image that has been corrupted by camera-shake. To the best of our knowledge this has not previously been done.

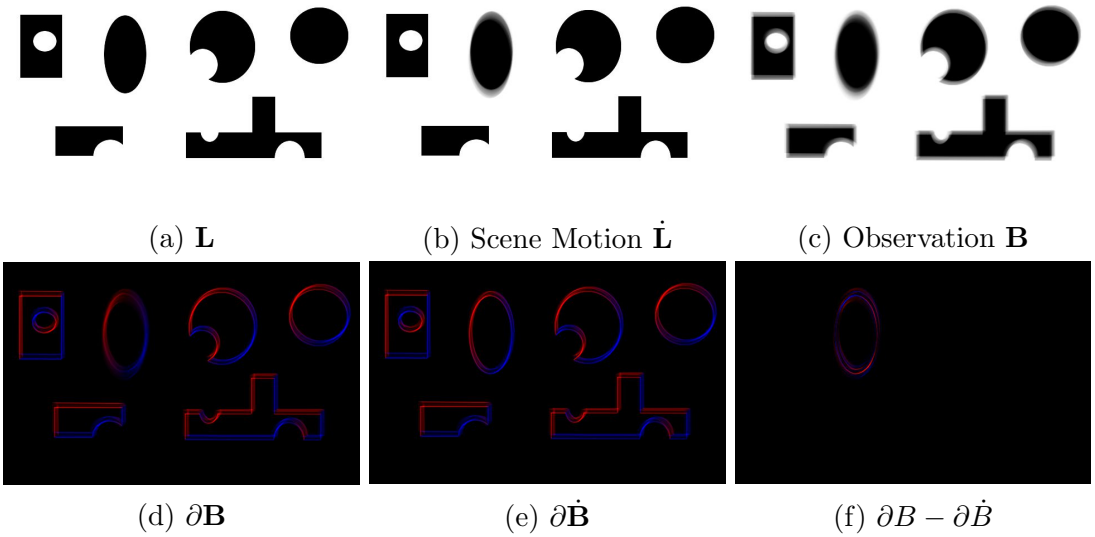
5.2.1 Understanding Scene Motion

An estimate of camera motion provided by inertial sensors gives a clue as to how the static elements of an image were blurred. This estimate is generally not sufficient to deconvolve the image, even if its content is static, as minor deviations in motion estimate caused by sensor noise can alter the blur function significantly. Joshi *et al.* [3] investigate the use of inertial sensors to aid static-scene blind deblurring, and assume the orientation estimate provided by gyroscope data to be accurate, and search for an optimal translating motion that best explains the blurred observation. Hirsch *et al.* [52] later presented improved results using just a 3DOF motion model, without the inertial data, on the same images used in [3], demonstrating the need to refine the inertial estimates.

5.2.2 Experiments

We attempted to find a per-pixel error between the gradients in the blurred image and the gradients in the re-blurred image. For a given blurry image $\mathbf{B}$, corrupted by both scene motion and camera shake, there is the instantaneous unblurred latent image $\mathbf{L}$. If we assume that this latent image is known, as well as the camera motion $\mathbf{A}$ that caused the global blur, then we can reconstruct the blurry image:

$$\dot{\mathbf{B}} = \mathbf{L} * \mathbf{A} \tag{5.17}$$



(a) $\mathbf{L}$ (b) Scene Motion $\dot{\mathbf{L}}$ (c) Observation $\mathbf{B}$

(d) $\partial\mathbf{B}$ (e) $\partial\dot{\mathbf{B}}$ (f) $\partial B - \partial\dot{B}$

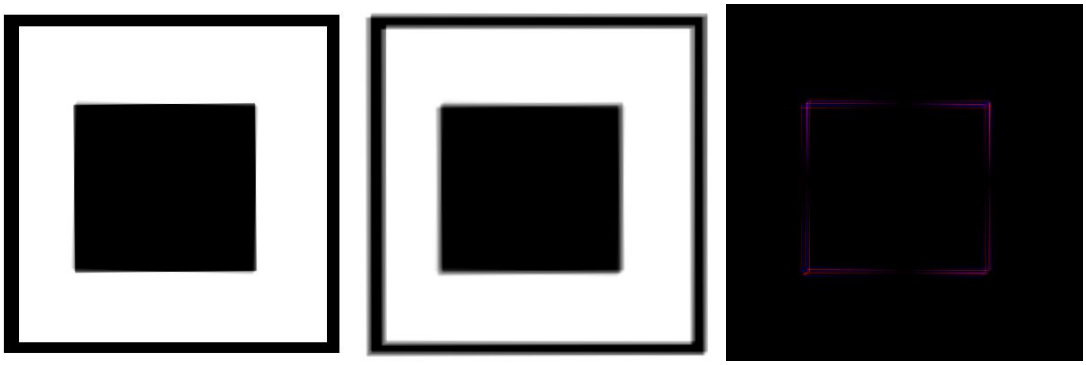
Figure 5-6: Finding blur due to scene motion in a synthetic example. (a) is the latent image. (b) shows the blurred image that would have been captured had the camera been static. (c) is the blurred observation captured by the moving camera, and (d) the corresponding image gradients. (e) is the gradients in $\mathbf{L}$ after applying the global camera motion. (f) is the difference between (d) and (e), indicating scene motion. The gradient images are colour-coded with red indicating a positive gradient and blue a negative gradient.

In ideal conditions this result would be equal to $\mathbf{B}$ for static regions, with any differences attributed to scene motion. This process is outlined in the synthetic example shown in figure 5-6. Here, we see that the gradients in the moving ellipse are different to the expected gradients according to the global camera motion blur.

Figure 5-7 shows that even with perfect synthetic data, the grouping differences between the observed image and the predicted blur is challenging. Depending on the motion, the differences can be minimal at many locations. This implies that a scheme for segmenting independent scene motion must consider the global fit of candidate objects in order to assess the correlation with camera motion.

Before we could perform this comparison on a real blurry image we had to tackle the issue of how we find $\dot{\mathbf{B}}$, which is dependent on the latent image and the camera motion. We used the motion according to the inertial sensors to deconvolve $\dot{\mathbf{B}}$ and then found the sharp prediction $\hat{\mathbf{L}}$ using the bilateral and shock filter technique discussed previously.

We compared the gradients in real blurry images with the predicted gradients



(a) Scene Motion $\dot{L}$ (b) Observed blur B (c) $\partial B - \partial \dot{B}$

Figure 5-7: A synthetic example demonstrating the difficulties of finding salient moving regions from a blurry image. The square in (a) is moving independently of the background lines. (b) shows the blurred observation, with the lines at the edges now blurred by the global camera motion. (c) shows the difference between the observation in (b) and the result of blurring (a) according to the global blur function. The differences are only apparent in the corners of the square.

($\partial\dot{\mathbf{B}}$). The difference between the observed blur gradients and the predicted blur gradients was as much the result of inaccuracies in the sharp prediction as it was a result of independent motion. Whilst the sharp prediction method can work well over large areas, where the errors in $\hat{\mathbf{L}}$ cancel out, it is not suitable for detecting motion of individual pixels.

Local Blur Functions

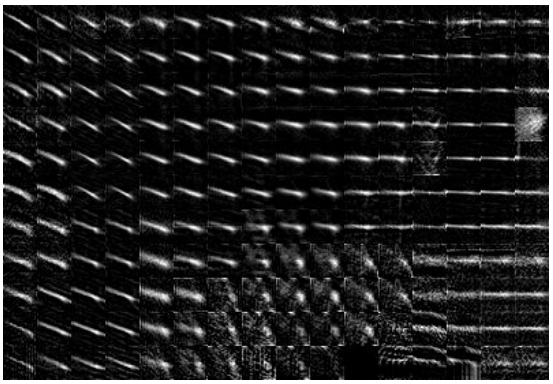
The camera motion estimate from the inertial sensors can give us an estimated blur kernel at each pixel of the image. Motion can be detected by comparing this estimate with the observed blur function of small windows. We estimated the blur function in overlapping windows using the fast method of [55], with the observed blurred image $\mathbf{B}$ and the sharp prediction $\hat{\mathbf{L}}$. The latter was computed on the result of deconvolving $\mathbf{B}$ according to the inertial motion estimate.

The result of this method is shown in Figure 5-8. The book in the foreground is moving, and has blur kernels that differ from the sensor predictions. Some regions of the image have errors in the kernel estimation as a result of not containing suitable features for estimating a blur function.

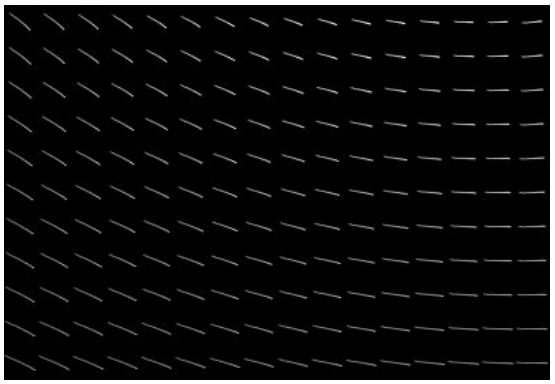
Comparing the kernels estimated from the small windows with the blur kernels from the inertial sensors is not straightforward. Whilst it is visibly apparent that

(a) $\mathbf{B}$

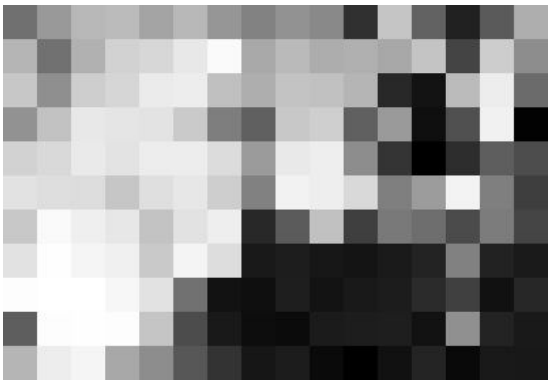
(b) $\hat{\mathbf{L}}$



(c) Observed blur kernels



(d) Inertial prediction of blur kernels



(e) Motion Correlation

Figure 5-8: (a) is an image corrupted by camera-shake and scene motion (the foreground book in lower-centre of image). (b) is the result of deconvolving (a) using the blur kernels estimated by sensor data to get an estimate of the unblurred image, ignoring any possible scene motion. Here we see that the background has been sharpened whilst the moving book remains blurred. (c) shows the estimated blur kernels for each region in the image, found as the blur function that corrupts $\hat{\mathbf{L}}$ to make $\mathbf{B}$. (d) is the sensor-predicted blur kernels, which are very similar to those in (c), apart from in the moving regions. (e) shows the correlation between the observed motion and predicted motion, with significant error in the bottom part of the image, corresponding to the motion of the book.

many of the static regions were blurred in a way that was *similar* to the estimated kernels, small errors due to noise cause a naive comparison such as sum-of-squared-differences to rate the kernels as being very different.

Instead of directly comparing the predicted kernels with the observed kernels, we fit a 2D Gaussian to each and find the direction and magnitude. The correlation in Figure 5-8e is found using the motion correlation method described in Chapter 2. Because the blur information is inferred from patches, our correlation is very coarse, but still manages to serve as an indicator of scene motion. The kernels corresponding to the image regions containing the book have low correlation compared to most of the rest of the image which has a high correlation.

5.3 Conclusion

Image deblurring remains a very open topic. The current state of the art can produce some excellent results when the assumptions are met. We found that these assumptions are difficult to achieve in practise with a blurred image captured by a real camera.

The practical implementation of deblurring methods is error-prone due to the sensitivity of the methods to parameters. Our belief is that inertial sensors can improve this situation by giving much-needed assistance in this ill-posed task. We have shown the potential of using inertial sensors to identify independent scene motion, which is a contribution as there exists no methods in the literature for segmenting a single blurred image. The next logical step would be to incorporate the correlation grid in Figure 5-8e with the motion segmentation scheme we've presented in Chapter 4. The r-map of [56] could be used in place of optical flow confidence to identify which areas of the correlation image contain reliable information.

The data provided by inertial sensors can assist in both determining the magnitude of camera motion as well as deciding which poses are unlikely to have contributed towards the blurred image. This can be used to limit the size of the pose space, ultimately reducing the computation required to deblur an image.

In the overall context of this thesis, this chapter has presented another application of inertial sensors. We've described the difficulties and problems associated with image deblurring, and shown that inertial sensor data can provide a vital cue in this ill-posed problem. Whilst this is certainly a rough result, it demonstrates the

utility of the sensor data and could provide the basis for a more complete method.

Chapter 6

Discussion

This thesis explored how the data provided by inexpensive inertial sensors can be used to better understand motion in several computer vision problems. In this chapter we summarise our contributions and assert how they support our hypothesis.

We began in Chapter 2 by describing the theory behind inertial sensors and how the motion they capture can be related to the motion in an image. This chapter also included details of the experimental setup and design of the embedded inertial sensing platform. As part of this design we developed a novel method for aligning recorded sensor data with captured video, using dense motion data and a GPU implementation to find the global minimum solution. This compared favourably to the literature, as we showed that our use of dense motion produced results that were consistently accurate, whereas methods using sparse features had failure modes, such as in scenes with few distinct features.

Our first contribution in support of our hypothesis was the development of a learned optical flow confidence measure that was trained using inertial sensor data. Our quantitative evaluation showed that a confidence measure trained using real-images out performed one trained using synthetic data when used to predict confidence for real images. Additionally we showed a performance benefit when using a confidence measure trained on images of the environment in which it is to be used. This is where the benefit of inertial sensors becomes most apparent, as the data could be used to build a confidence measure that simultaneously learns and adapts to its environment. Ultimately we showed that inertial sensor data can simplify the understanding of optical flow fields by being used as the basis for training a confidence measure.

The primary contribution of this thesis is a real-time method for segmenting non-rigid motion from a single pair of images. This was made possible for two reasons. Firstly, our novel use of an optical flow confidence measure trained on similar scenes meant that we were able to use cheap-to-produce, noisy optical flow fields. This is in contrast to the state of the art motion segmentation scheme, which has a significant reduction in segmentation accuracy when using cheap optical flow data. Secondly, when this was combined with an estimate of motion from the inertial sensors, we were able to quickly identify regions that were likely to have been moving independently. This information was then combined with standard image segmentation techniques to produce a final segmentation result. Our evaluation showed a total run time reduction by an average factor of nearly 50 when compared to the state of the art in fast video segmentation for an equivalent accuracy, all whilst using the information from just a single pair of frames.

We compared our inertial method with one using motion inferred from the images and found that, whilst in most cases it offered an equivalent accuracy, it ran at an average of half the speed of the inertial method, showed great variability in run time and had significant failure modes in certain scenes. By using inertial sensors a complex and error-prone step could be removed and replaced with a very simple estimation of motion.

The method we presented is ideally suited to real-time applications. The bulk of the optical flow confidence estimation and motion analysis is inherently parallel and simple to implement. The time taken to process each frame is consistent, and our use of a learned optical flow confidence measure means we can adapt to varying flow field quality, depending on the available hardware resources. These are all direct benefits of using inertial sensor data.

Finally we looked at the topic of motion blur. After describing the various problems associated with estimating motion from blur, as well as the effect of scene motion, we showed how the inertial sensor data can be used to differentiate between blur caused solely by camera motion and blur caused by a combination of both camera and scene motion. Whilst this aspect of our work is still in the preliminary stages, it demonstrates that the estimated dynamics of a moving camera from inertial sensors can be compared with observed blur kernels to identify independent motion. With continued development, this can lead to understanding and reversing motion blur in blurry images containing scene motion which is not handled by current

techniques.

The methods we've developed are applicable to current consumer mobile devices that contain the essential components; camera and sensors. These sensors are already starting to be used for imaging tasks, such as delaying shutter actuation until the camera is stationary, to avoid taking a blurry image. It is in these scenarios, where limited resources are available, that the techniques we've developed have the most relevance.

Bibliography

[1] N. Trawny and S. I. Roumeliotis, "Indirect Kalman filter for 3D attitude estimation," Tech. Rep. 2005–002, University of Minnesota, Dept. of Comp. Sci. & Eng., Mar. 2005.

[2] R. Köhler, M. Hirsch, B. Mohler, B. Schölkopf, and S. Harmeling, "Recording and playback of camera shake: Benchmarking blind deconvolution with a real-world database," in *Proceedings of the 12th European Conference on Computer Vision - Volume Part VII*, ECCV'12, pp. 27–40, 2012.

[3] N. Joshi, S. B. Kang, C. L. Zitnick, and R. Szeliski, "Image deblurring using inertial measurement sensors," *ACM Trans. Graph.*, vol. 29, pp. 30:1–30:9, July 2010.

[4] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[5] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, (San Francisco, CA, USA), pp. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[6] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *ARTIFICAL INTELLIGENCE*, vol. 17, pp. 185–203, Aug. 1981.

[7] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Proceedings of the 13th Scandinavian Conference on Image Analysis*, LNCS 2749, (Gothenburg, Sweden), pp. 363–370, June-July 2003.

[8] S. Madgwick, A. Harrison, and R. Vaidyanathan, "Estimation of imu and marg orientation using a gradient descent algorithm," in *Rehabilitation Robotics*

(ICORR), 2011 IEEE International Conference on, pp. 1–7, 29 2011-july 1 2011.

[9] M. Aron, G. Simon, and M.-O. Berger, "Handling uncertain sensor data in vision-based camera tracking," in *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '04, (Washington, DC, USA), pp. 58–67, IEEE Computer Society, 2004.

[10] M. Hwangbo, J.-S. Kim, and T. Kanade, "Inertial-aided klt feature tracking for a moving camera," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 1909–1916, Oct. 2009.

[11] M. Hwangbo, J.-S. Kim, and T. Kanade, "Gyro-aided feature tracking for a moving camera: fusion, auto-calibration and gpu implementation," *The International Journal of Robotics Research*, vol. 30, pp. 1755–1774, Dec. 2011.

[12] A. Karpenko, D. Jacobs, J. Baek, and M. Levoy, "Digital video stabilization and rolling shutter correction using gyroscopes," *Stanford Tech Report CTSR 2011-03*.

[13] D. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, pp. 1150–1157 vol.2, 1999.

[14] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, pp. 679–698, June 1986.

[15] S. Meister and D. Kondermann, "Real versus realistically rendered scenes for optical flow evaluation," in *Electronic Media Technology (CEMT), 2011 14th ITG Conference on*, pp. 1–6, 2011.

[16] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *Int. Journal Computer Vision*, vol. 92, pp. 1–31, Mar. 2011.

[17] J. Barron, D. Fleet, S. Beauchemin, and T. A. Burkitt, "Performance of optical flow techniques," in *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on*, pp. 236–242, June 1992.

[18] A. Bainbridge-Smith and R. Lane, "Measuring confidence in optical flow estimation," *Electronics Letters*, vol. 32, pp. 882–884, May 1996.

[19] P. Mrquez-Valle, D. Gil, A. Hernndez-Sabat, and D. Kondermann, "When is a confidence measure good enough?," in *Computer Vision Systems* (M. Chen, B. Leibe, and B. Neumann, eds.), vol. 7963 of *Lecture Notes in Computer Science*, pp. 344–353, Springer Berlin Heidelberg, 2013.

[20] S. Uras, F. Girosi, A. Verri, and V. Torre, "A computational approach to motion perception," *Biological Cybernetics*, vol. 60, no. 2, pp. 79–87, 1988.

[21] E. Simoncelli, E. Adelson, and D. Heeger, "Probability distributions of optical flow," in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, pp. 310–315, Jun.

[22] B. Jahne and P. Geissler, eds., *Handbook of Computer Vision and Applications*, vol. 2. Academic Press, 1st ed., 1999.

[23] A. Bruhn and J. Weickert, "A confidence measure for variational optic flow methods," in *Geometric Properties for Incomplete data* (R. Klette, R. Kozera, L. Noakes, and J. Weickert, eds.), vol. 31, pp. 283–298, Springer Netherlands, 2006.

[24] J. Kybic and C. Nieuwenhuis, "Bootstrap optical flow confidence and uncertainty measure," *Computer Vision and Image Understanding*, vol. 115, no. 10, pp. 1449 – 1462, 2011.

[25] C. Kondermann, R. Mester, and C. Garbe, "A statistical confidence measure for optical flows," in *10th European Conference on Computer Vision*, pp. 290–301, Springer Berlin Heidelberg, Oct. 2008.

[26] O. Mac Aodha, A. Humayun, M. Pollefeys, and G. J. Brostow, "Learning a confidence measure for optical flow," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[27] L. B. Statistics and L. Breiman, "Random forests," in *Machine Learning*, pp. 5–32, 2001.

[28] B. Catanzaro, B.-Y. Su, N. Sundaram, Y. Lee, M. Murphy, and K. Keutzer, "Efficient, high-quality image contour detection," in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 2381–2388, Oct. 2009.

[29] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[30] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning*, ICML '06, (New York, NY, USA), pp. 161–168, ACM, 2006.

[31] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 5, pp. 603–619, 2002.

[32] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.

[33] Y. Boykov and G. Funka-Lea, "Graph cuts and efficient n-d image segmentation," *Int. Journal Computer Vision*, vol. 70, pp. 109–131, Nov. 2006.

[34] C. Nieuwenhuis, B. Berkels, M. Rumpf, and D. Cremers, "Interactive motion segmentation," in *Proceedings of the 32nd DAGM conference on Pattern recognition*, (Berlin, Heidelberg), pp. 483–492, Springer-Verlag, 2010.

[35] D. Zhou, L. Wang, X. Cai, and Y. Liu, "Detection of moving targets with a moving camera," in *Robotics and Biomimetics (ROBIO), 2009 IEEE International Conference on*, pp. 677–681, 2009.

[36] T. Brox and J. Malik, "Object segmentation by long term analysis of point trajectories," in *Computer Vision ECCV 2010* (K. Daniilidis, P. Maragos, and N. Paragios, eds.), vol. 6315 of *Lecture Notes in Computer Science*, pp. 282–295, Springer Berlin Heidelberg, Sept. 2010.

[37] P.Ochs and T.Brox, "Higher order motion models and spectral clustering," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[38] D. Cremers and S. Soatto, "Motion competition: A variational approach to piecewise parametric motion segmentation," *Int. J. Comput. Vision*, vol. 62, pp. 249–265, May 2005.

[39] T. Dinh and G. Medioni, "Two-frames accurate motion segmentation using tensor voting and graph-cuts," *Motion and Video Computing, IEEE Workshop on*, vol. 0, pp. 1–8, 2008.

[40] C. Xu, J. Liu, and B. Kuipers, "Moving object segmentation using motor signals," in *European Conference on Computer Vision (ECCV 2012)*, 2012.

[41] M. Unger, M. Werlberger, T. Pock, and H. Bischof, "Joint motion estimation and segmentation of complex scenes with label costs and occlusion modeling," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 1878–1885, June.

[42] A. Papazoglou and V. Ferrari, "Fast object segmentation in unconstrained video," *Computer Vision, IEEE International Conference on*, vol. 0, pp. 1777–1784, 2013.

[43] J. Lobo, J. Ferreira, P. Trindade, and J. Dias, "Bayesian 3d independent motion segmentation with imu-aided rbg-d sensor," in *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012 IEEE Conference on*, pp. 445–450, Sept.

[44] C. Y. Ren and I. Reid, "gSLIC: a real-time implementation of SLIC superpixel segmentation," *Oxford University Technical Report*, 2011.

[45] A. Delong, A. Osokin, H. Isack, and Y. Boykov, "Fast approximate energy minimization with label costs," *International Journal of Computer Vision*, vol. 96, no. 1, pp. 1–27, 2012.

[46] C. Rother, V. Kolmogorov, and A. Blake, ""grabcut": Interactive foreground extraction using iterated graph cuts," *ACM Trans. Graph.*, vol. 23, pp. 309–314, Aug. 2004.

[47] T. Brox and J. Malik, "Large displacement optical flow: descriptor matching in variational motion estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 3, pp. 500–513, 2011.

[48] R. Fergus, B. Singh, A. Hertzmann, S. T. Roweis, and W. T. Freeman, "Removing camera shake from a single photograph," *ACM Trans. Graph.*, vol. 25, pp. 787–794, July 2006.

[49] O. Whyte, J. Sivic, A. Zisserman, and J. Ponce, "Non-uniform deblurring for shaken images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[50] M. Hirsch, S. Sra, B. Schlkopf, and S. Harmeling in *Efficient Filter Flow for Space-Variant Multiframe Blind Deconvolution*, pp. 607–614, June 2010.

[51] A. Gupta, N. Joshi, C. L. Zitnick, M. Cohen, and B. Curless, "Single image deblurring using motion density functions," in *Proceedings of the 11th European conference on Computer vision*, ECCV'10, pp. 171–184, 2010.

[52] M. Hirsch, C. Schuler, S. Harmeling, and B. Schölkopf, "Fast removal of non-uniform camera shake," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 1–8, IEEE, Nov. 2011.

[53] Z. Hu and M. hsuan Yang, "Fast non-uniform deblurring using constrained camera pose subspace," in *Proceedings of the British Machine Vision Conference*, pp. 136.1–136.11, BMVA Press, Sept. 2012.

[54] Q. Shan, J. Jia, and A. Agarwala, "High-quality motion deblurring from a single image," *ACM Transactions on Graphics (SIGGRAPH)*, 2008.

[55] S. Cho and S. Lee, "Fast motion deblurring," in *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, (New York, NY, USA), pp. 145:1–145:8, ACM, 2009.

[56] L. Xu and J. Jia, "Two-phase kernel estimation for robust motion deblurring," in *Proceedings of the 11th European conference on Computer vision: Part I*, ECCV'10, (Berlin, Heidelberg), pp. 157–170, Springer-Verlag, 2010.

[57] D. Krishnan and R. Fergus, "Fast Image Deconvolution using Hyper-Laplacian Priors," in *Advances in Neural Information Processing Systems 22* (Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, eds.), pp. 1033–1041, 2009.

[58] N. Joshi, R. Szeliski, and D. Kriegman, "Psf estimation using sharp edge prediction," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, June 2008.

[59] O. Whyte, J. Sivic, and A. Zisserman, "Deblurring shaken and partially saturated images," in *Proceedings of the IEEE Workshop on Color and Photometry in Computer Vision, with ICCV 2011*, 2011.

[60] L. Xu and J. Jia, "Depth-aware motion deblurring," in *Computational Photography (ICCP), 2012 IEEE International Conference on*, pp. 1–8, 2012.

[61] L. Sun, S. Cho, J. Wang, and J. Hays, "Edge-based blur kernel estimation using patch priors," in *Proc. IEEE International Conference on Computational Photography*, 2013.

[62] S. Osher and L. I. Rudin, "Feature-oriented image enhancement using shock filters," *SIAM J. Numer. Anal.*, vol. 27, pp. 919–940, Aug. 1990.

[63] S. Cho, J. Wang, and S. Lee, "Handling outliers in non-blind image deconvolution," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 495–502, 2011.

[64] A. Levin, "Blind motion deblurring using image statistics," in *In Advances in Neural Information Processing Systems*, 2006.

[65] A. Chakrabarti, T. Zickler, and W. Freeman, "Analyzing spatially-varying blur," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2512–2519, 2010.

[66] K. Bahrami, A. C. Kot, and J. Fan, "A novel approach for partial blur detection and segmentation," in *Multimedia and Expo (ICME), 2013 IEEE International Conference on*, pp. 1–6, July 2013.

[67] S. Cho, Y. Matsushita, and S. Lee, "Removing non-uniform motion blur from images," *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, Oct. 2007.